POLITECNICO DI TORINO

Master of Science in Electronics Engineering

Master Thesis

Analisi di strumenti Open-source per la progettazione di circuiti integrati

Politecnico di Torino

Supervisors
prof. Mario Roberto Casu

Candidato
Antonino Magaddino

Data di Laurea

Abstract

Opensource has developed considerably in recent years. The world of electronics has been fully influenced by it. Examples are the birth of the concept of Open Source Hardware (OSH) or projects aimed at the creation and improvement of software, without license, able to support the electronic engineer during the different steps of his work.
This thesis fits into that context. We tried to investigate the possibility of using a design flow from RTL to GDS completely open source (OpenROAD project) and we made a comparison between the results obtained from this flow with those produced by proprietary software (Synopsys Design Compiler and Cadence Innovus).
The architecture taken as a model was the NVIDIA opensource deep learning accelerator (NVDLA), while the reference technology is the SKY130, opensource PDK (Process Design Kit) born from the collaboration between the foundry SkyWater and Google.
The analysis work was divided into two parts:

- Synthesis results comparison;
- P&R results comparison.

Regarding synthesis we observed and compared, in terms of timing, power and design area, the netlists generated by Yosys (opensource synthesis tool) and by Synopsys Design Compiler (proprietary tool), starting from the verilog source code of the architecture.
Similar work was also conducted for the P&R part. In this case we found more criticalities on the "opensource" side, since the excessive complexity of the architecture proved unmanageable by the software of the OpenROAD flow. We have, therefore, decided to divide the NVIDIA accelerator into its elementary components and conduct the investigation only on those analyzable by opensource tools. We finally compared the results with those generated by the Innovus-licensed tool.

Sommario

Negli ultimi anni l'Opensource ha avuto un considerevole sviluppo. Il mondo dell'elettronica ne ha subito pienamente l'influsso. Ne sono esempio la nascita del concetto di Open Source Hardware (OSH) o di progetti volti alla creazione e al perfezionamento di software, privi di licenza, in grado di affiancare l'ingegnere elettronico durante i diversi passi del suo lavoro.
Questa tesi si inserisce in tale contesto. Abbiamo provato ad indagare la possibilità di utilizzo di un flusso di progetto da RTL a GDS completamente open source (progetto OpenROAD) e abbiamo effettuato un confronto tra i risultati ottenibili da questo flusso con quelli prodotti da software proprietari (Synopsys Design Compiler e Cadence Innovus).
L'architettura presa a modello è stata l'acceleratore di deep learning opensource di NVIDIA (NVDLA – NVIDIA Deep Learning Accelerator), mentre la tecnologia di riferimento è la SKY130, PDK (Process Design Kit) opensource nato dalla collaborazione tra la foundry SkyWater e Google.
Il lavoro di analisi è stato suddiviso in due parti:

- Confronto risultati di sintesi;
- Confronto risultati di P&R.

Per quanto riguarda la sintesi abbiamo osservato e comparato, in termini di timing, potenza e area del design, le netlist generate da Yosys (tool opensource di sintesi) e da Synopsys Design Compiler (tool proprietario), a partire dal codice verilog sorgente dell'architettura.
Un lavoro simile è stato condotto anche per la parte di P&R. In questo caso sono state riscontrate più criticità per quel che riguarda il lato "opensource" in quanto l'eccessiva complessità dell'architettura si è dimostrata ingestibile da parte dei software del flusso OpenROAD. Abbiamo, quindi, deciso di dividere l'acceleratore NVIDIA nelle sue componenti elementari e condurre l'indagine solo su quelle analizzabili dagli strumenti opensource. Abbiamo infine comparato i risultati con quelli generati dal tool con licenza Innovus.

Indice

Elenco delle tabelle 6

Elenco delle figure 8

List of acronyms and abbreviations 11

1 Introduzione 14

2 Progetto OpenROAD: democratizzazione della progettazione hardware 16
2.1 Programma IDEA . . . 16
2.2 Progetto OpenROAD: flusso da RTL a GDSII opensource . . . 17
2.2.1 Sintesi logica: Yosys/ABC . . . 19
2.2.2 Floorplan e PDN (Power Delivery Network): TritonFP . . . 19
2.2.3 Placement . . . 21
2.2.4 CTS: Triton CTS . . . 24
2.2.5 Routing . . . 26
2.2.6 Static Timing Analysis: OpenSTA . . . 29

3 Architettura NVDLA: Acceleratore di Deep Learning 31
3.1 CNN: Reti Neurali Convoluzionali . . . 31
3.2 NVDLA: NVIDIA Deep Learning Accelerator . . . 32
3.2.1 Modelli verilog . . . 33
3.2.2 NVDLA_partition_a . . . 35

4 Sintesi 40
4.1 Analisi temporale . . . 42
4.2 Analisi di potenza . . . 48
4.3 Analisi dell'area . . . 52

5 P&R 55
5.1 Analisi temporale . . . 56
5.2 Analisi di area . . . 59
5.3 Analisi di potenza . . . 61

6 Conclusione: pro e contro del flusso OPENRoad 68
6.1 Potenzialità 68
6.2 Criticità 69

A NVDLA: script di configurazione 71

B NVDLA: codici verilog 81

C Critical path di sintesi e PR 82

Bibliografia 90

Elenco delle tabelle

4.1 Analisi temporale post-sintesi di netlist generate da Yosys/ABC nell'ambiente OpenROAD ad una frequenza operativa di $2MHz$ 43
4.2 Analisi post-sintesi di netlist generate da Synopsys Design Complier nell'ambiente Synopsys ad una frequenza operativa di $2MHz$ 43
4.3 Analisi post-sintesi di netlist generate da Yosys/ABC nell'ambiente Synopsys ad una frequenza operativa di $2MHz$ 44
4.4 Analisi temporale post-sintesi di netlist generate da Yosys/ABC nell'ambiente OpenROAD ad una frequenza operativa di $200MHz$ 44
4.5 Analisi post-sintesi di netlist generate da Synopsys Design Complier nell'ambiente Synopsys ad una frequenza operativa di $200MHz$ 45
4.6 Analisi post-sintesi di netlist generate da Yosys/ABC nell'ambiente Synopsys ad una frequenza operativa di $200MHz$ 45
4.7 Analisi post-sintesi di netlist generate da Yosys/ABC nell'ambiente OpenROAD ad una frequenza operativa di $2MHz$ 49
4.8 Analisi post-sintesi di netlist generate da Synopsys Design Compiler nell'ambiente Synopsys ad una frequenza operativa di $2MHz$ 49
4.9 Analisi post-sintesi di netlist generate da Yosys/ABC nell'ambiente Synopsys ad una frequenza operativa di $2MHz$ 50
4.10 Analisi post-sintesi di netlist generate da Yosys/ABC nell'ambiente OpenROAD ad una frequenza operativa di $200MHz$ 51
4.11 Analisi post-sintesi di netlist generate da Synopsys Design Compiler nell'ambiente Synopsys ad una frequenza operativa di $200MHz$ 51
4.12 Analisi post-sintesi di netlist generate da Yosys/ABC nell'ambiente Synopsys ad una frequenza operativa di $200MHz$ 52
4.13 Analisi post-sintesi delle aree degli elementi facenti parte della gerarchia di NV_NVDLA_partition_a 53
4.14 Analisi post-sintesi delle aree degli elementi facenti parte della gerarchia di NV_NVDLA_partition_a che utilizzano il blocco DW_lsd 53
4.15 Analisi post-sintesi delle aree degli elementi facenti parte della gerarchia di NV_NVDLA_partition_a che utilizzano il blocco NV_DW_lsd 54

5.1 Risultati di timing, in termini di setup e hold time, durante il P&R ad una frequenza operativa di 2MHz 58

5.2 Risultati di timing, in termini di setup e hold time, durante il P&R ad una frequenza operativa di 200MHz 58
5.3 Report delle aree occupate dalle istanze analizzate durante il P&R 61
5.4 Risultati di potenza totale dissipata dai blocchi analizzati durante il P&R ad una frequenza operativa di 2MHz 62
5.5 Risultati di potenza totale dissipata dai blocchi analizzati durante il P&R ad una frequenza operativa di 200MHz 65

Elenco delle figure

2.1 Crisi tecnologica del design [1] . . . 17
2.2 Flusso RTL-GDSII di OpenROAD [1 . . . 18
2.3 Differenti livelli di astrazione e di sintesi [2] . . . 19
2.4 Floorplan e PDN del NDVLA_reset . . . 20
2.5 (a)Illustrazione di una sequenza verticale di una rete. (b) Potenziali soluzioni di routing ottimale per la reta in (a) [5] . . . 22
2.6 (a)Risultato post global placement per NVDLA_reset, con celle standard evidenziate. (b) Rappresentazione completa post-placement . . . 23
2.7 (a)Risultato post global placement per NVDLA_reset, con celle standard evidenziate. (b) Risultato post detailed placement per NVDLA_reset, con celle standard evidenziate . . . 24
2.8 Esempio di GH-tree [4] . . . 24
2.9 (a)Risultato post detailed placement per NVDLA_reset, con celle standard evidenziate. (b) Risultato post CTS per NVDLA_reset, con celle standard evidenziate . . . 25
2.10 Risultato post-CTS e post inserimento FILLER cells per NVDLA_reset . 26
2.11 Flow FastRoute [6] . . . 27
2.12 Un esempio di "route guide" per una rete a quattro pin: (a) vista dall'alto 2D e (b) vista 3D [7] . . . 28
2.13 Un esempio di "timing paths" . . . 29

3.1 Esempio degli strati di una CNN[8] . . . 32
3.2 Architettura NVDLA[8] . . . 33
3.3 Partizioni NVDLA [9] . . . 35
3.4 Pipeline di convoluzione[8] . . . 36
3.5 Struttura interna CACC[8] . . . 36
3.6 Layout di una SRAM a 2 banchi, da 16kbyte implementata su OpenRAM [10] . . . 37
3.7 Organizzazione framework OpenRAM [11] . . . 38
3.8 Struttura interna banco RAM su OpenRAM [11] . . . 39

4.1 Gerarachia NVDLA_partition_a . . . 41

5.1 Celle utilizzate da Cadence Innovus Implementation System post-CTS di NV_NVDLA_reset . . . 57

5.2 Celle utilizzate da TritonCTS post-CTS di NV_NVDLA_reset 57
5.3 Dettaglio delle celle filler inserite nel flusso OPERNRoad di NV_NVDLA_reset 59
5.4 Dettaglio delle celle filler inserite nel flusso Cadence Innovus Implementation System NV_NVDLA_reset 60
5.5 sky130_fd_sc_hd___buf_1 64
5.6 sky130_fd_sc_hd___clkinv_16 64
5.7 Risultato finale su OPENRoad di NV_NVDLA_reset 66
5.8 Risultato finale su Cadence Innovus Implementation System di NV_NVDLA_reset 66

List of acronyms and abbreviations

AI	Artificial Intelligence
ANN	Artificial Neural Network
APB	Advanced Peripheral Bus
ARM	Advanced RISC Machines Ltd
ASCII	American Standard Code for Information Interchange
AXI	Advanced eXtensible Interface
BSD	Berkeley Source Distribution
CACC	Convolution Accumulator
CBUF	Convolution Buffer
CDMA	Convvolution Direct Memory Access
CMAC	Convvolution Multiply-Accumulation
CNN	Convolutional Neural Network
CPS	Critical Path Slack
CSB	Configuration Space Bus
CSC	Convolution Sequence Controller
CTS	Clock Tree Synthesis
DARPA	Defence Advance Research Project Agency
DEF	Design Exchange Format
DRC	Design Rule Check
DRV	Design Rule Violation
EDA	Electronic Design Automation
ERI	Electronics Resurgence Initiative
EOL	End-Of-Line
FOSS	Free and Open-Source Software
GCELL	Global Routing Cell
GDS	Graphic Design System
HDL	Hardware Description Language
IDEA	Intelligent Design of Electronic Assets
IC	Integrated Circuit
IP	Intellectual Property
ISPD	International Symposium on Physical Design
LEF	Library Exchange Format
MILP	Mixed Integer-Linear Programming
NVDLA	NVIDIA Deep Learning Accelerator
NPU	Neural Processing Unit
OPENDP	Open-Source Detailed Placement Engine
OPENROAD	(Realization of Open, Accessible Design
OSH	Open-Source Hardware
PDK	Process Design Kit
PDN	Power Delivery Network

P&R	Place and Route
POWV	Potential Optimal Wirelength Vector
RAM	Random Access Memory
RTL	Register-Transfer Level
SA	Simulated Annealing
SDC	Synopsys Design Constraint
SDF	Standard Delay Format
SOC	System-on-Chip
SPEF	Standard Parasitic Exchange Format
STA	Static Timing Analysis
TCL	Tool Command Language
TLM	Transaction Level Model
TNS	Total Negative Slack
YOSYS	Yosys Open SYnthesis Suite
UFRGS	Università federale del Rio Grande do Sul
WNS	Worst Negative Slack

Capitolo 1

Introduzione

E' possibile utilizzare un flusso di progetto da RTL a GDSII completamente open-source? Ma soprattutto, si è in grado di ottenere risulati se non migliori quantomeno confrontabili con l'insieme di strumenti ad oggi usati dalle aziende di alta tecnologia quali ARM, Qualcomm, etc...?
Queste sono le domande a cui il programma IDEA (Intelligent Design of Electronic Assets) del DARPA (Defence Advance Research Project Agency) ha provato a rispondere quando, nel Giugno del 2018, ha lanciato il progetto OpenROAD (Foundations and Realization of Open, Accessible Design).
Il tentativo è quello di intraprendere una campagna di "democratizzazione della progettazione hardware" provando a far fronte alle molteplici barriere all'ingresso di cui l'industria dei seminconduttori è vittima (costi, tempi, competenze e imprevedibilità).
Abbinando tale progetto al concetto dell'OSH (Open-Source Hardware), ovvero l'insieme di design hardware privi di licenza e facilmente accessibili da chiunque, si protrebbe assistere, potenzialmente, ad una rivoluzione nel contesto della progettazione IC e non solo.
Questa tesi si incentra sulla possibiltà di uso del flusso OpenROAD per l'implementazione (sinstesi e P&R) di un'architettura di discreta complessità: una variante dell'acceleratore per deep learning open source di NVIDIA, NVDLA (NVIDIA Deep Learning Accelerator).
La prima parte sarà dedicata ad un'analisi concisa ed esaustiva del progetto OpenROAD. Successivamente saranno descritti in modo sommario gli strumenti facenti parte del flusso da RTL a GDSII, da Yosys+ABC per la sintesi, a RePlAce per il placement, fino a TritonCTS e TritonRoute per la sintesi dell'albero di clock (CTS) e per il ruoting.
A questo primo blocco seguirà un'esame dell'architettura NVIDIA-NVDLA, focalizzata su: principio di funzionamento e organizzazione interna della totalità dell'acceleratore.
Finita questa introduzione generale si volgerà lo sguardo verso una delle partizioni di cui NVDLA è composto, la partition_a, sulla quale ci si è dovuti concentrare data tutta una serie di limiti e difficoltà incontrate durante il lavoro di ricerca. Se ne analizzerà il compito e la struttura del codice sorgente. A tal proposito verrà presentato un accenno riguardante il software open-source OpenRAM, quadro di sviluppo di compilatori per RAM, poichè il NVDLA è di per sè sprovvisto di RAM sintetizzabili. OpenRAM

potrebbe, quindi, risultare un ottimo strumento da affiancare ad un simile lavoro di implementazione.
Terminata questa prima parte si passerà al corpo centrale della tesi: la comparazione dei risultati di sintesi e P&R, in termini di timing, potenza ed area, ottenuti sia usando gli strumenti del flusso OpenROAD sia i software proprietari, ovvero Synopsys Design Compiler (per la sintesi) e Innovus (per il P&R).
Si accenna in questa introduzione che per quel che riguarda il P&R ci si è dovuti limitare all'analisi dei componenti più semplici poichè il flusso OpenROAD non è stato in grado di arrivare a conclusione per i blocchi di complessità maggiore, compresa l'intera partizione_a.
Verranno infine presentate quelle che sono state le maggiori criticità incontrate durante il lavoro di implementazione, sia in termini di impossibilità nel procedere in determinati punti del flusso OpenROAD sia in difficoltà di reperimento di materiale utile al fine di poter superare tali situazioni di stallo.
In conclusione si commenteranno i possibili sviluppi futuri di un progetto che gode di alte potenzialità ma che soffre dell'assenza di un vero lavoro di ricerca e sviluppo atto ad affrontare con successo i vari limiti presenti, che lo rendono non ancora in grado di competere in modo soddisfacente con l'attuale stato dell'arte in termini di software per implementazione fisica di IC.

Capitolo 2

Progetto OpenROAD: democratizzazione della progettazione hardware

2.1 Programma IDEA

Nel giugno del 2018, durante il vertice ERI (Electronics Resurgence Initiative) del DARPA, venne presentato ufficialmente il programma IDEA.
In un contesto nel quale la complessità dei SoC (System-on-Chip), in termini di numero di transistor utilizzati e di interconnessioni tra gli stessi, è in una fase di aumento quasi esponenziale, le barriere all'ingresso che una qualsiasi realtà di progettazione hardware si trova a dover affrontare limitano in modo consistente le possibilità di lavoro.
Dopo essersi resi conto che, come la figura 2.1 può far intuire, «L'innovazione del sistema hardware è bloccata in un minimo locale di (i) strumenti complessi e costosi, (ii) una carenza di utenti esperti in grado di usare questi strumenti in tecnologie avanzate, e (iii) enormi barriere di costo e di rischio anche solo per tentare la progettazione dell'hardware»[1], è nata la necessità di ripensarla in modo da renderla il più accessibile possibile.

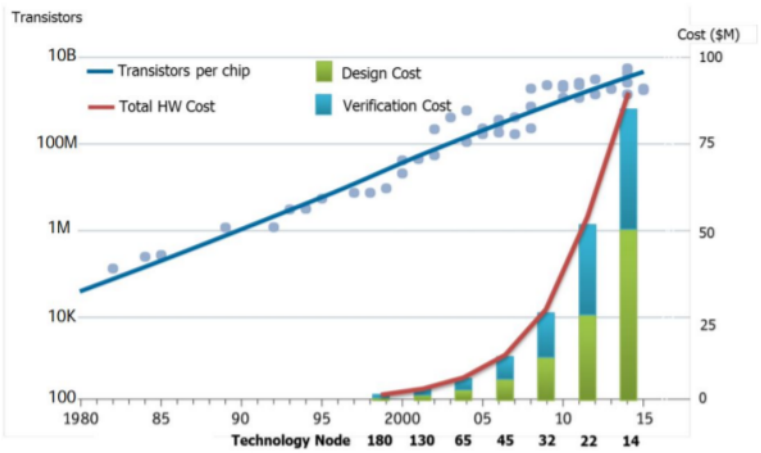


Figura 2.1: Crisi tecnologica del design [1]

Da questi presupposti nasce il programma del DARPA, IDEA, che «mira a sviluppare un generatore di layout di circuiti completamente automatizzato, "no human in the loop", che possa consentire agli utenti senza esperienza nella progettazione elettronica di completare la realizzazione fisica di hardware»[1].

2.2 Progetto OpenROAD: flusso da RTL a GDSII open-source

Sulla base di questi principi fondanti, il programma IDEA ha sviluppato il progetto OpenrROAD.
Si tratta di «un'insieme di strumenti, completamente autonomo e open-source, per la generazione di un layout digitale, a livello di die, package e scheda, con il focus iniziale sulla fase da RTL a GDSII della progettazione di un SoC»[1], si guardi la figura 2.2.

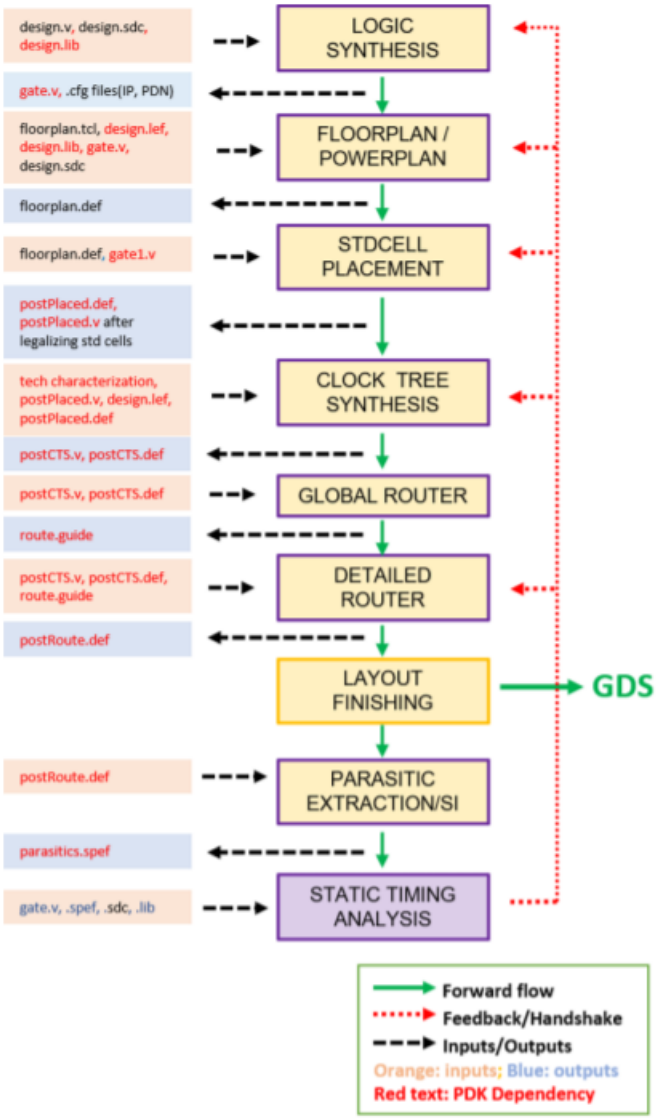


Figura 2.2: Flusso RTL-GDSII di OpenROAD [1

Alla base del progetto vi è la volontà di creare un «nuovo paradigma per quel che riguarda gli strumenti di EDA (Electronic Design Automation), la collaborazione tra ambiente industriale e accademico e la ricerca accademica stessa»[1]. Infatti l'avvio del progetto è stato reso possibile solo grazie alla convergenza di diversi elementi:

- Un'iniziale significativa IP (Intellectual Property) del software;
- Strumenti di STA (Static Timing Analysis) commerciali;
- Un insieme di IP di software di ambienti accademici con le correlate competenze;

- Le conoscenze in ambito SoC dei partner commerciali, quali Qualcomm e ARM;
- Un team di progettazione interno, proveniente dall'Università del Michigan

Definito il contesto in cui OpenROAD nasce, se ne analizzeranno adesso gli strumenti che ne fanno parte, dalla sintesi fino al routing.

2.2.1 Sintesi logica: Yosys/ABC

Il FOSS (Free and Open-Source Software) di sintesi, utilizzato nel flusso OpenROAD è Yosys (Yosys Open SYnthesis Suite). In realtà, nonostante Yosys integri le funzioni base di sintesi logica, esso sfrutta un altro software open-source, ovvero ABC, in grado di effettuare ottimizzazioni a livello di gate più avanzate.
«L'obiettivo del progetto ABC è quello di fornire un'implementazione di dominio pubblico degli algoritmi di sintesi combinatori e sequenziali allo stato dell'arte e, allo stesso tempo, creare un ambiente open-source in cui tali applicazioni possono essere sviluppate e confrontate»[2].
Sono entrambi strumenti di sintesi che lavorano con il HDL (Hardware Description Language) Verilog. Prendono come input le descrizioni comportamentali (.v) dei design e di generano, in output, le caratterizzazioni a livello RTL (Register-Transfer Level), a livello logico e fisico. In figura 2.3 viene mostrato il livello di astrazione in cui Yosys/ABC si inserisce.

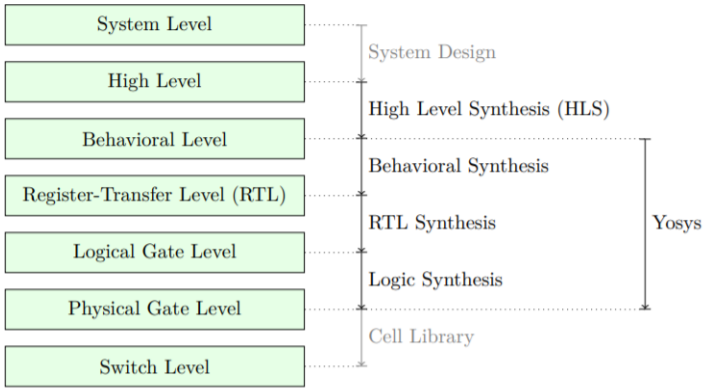


Figura 2.3: Differenti livelli di astrazione e di sintesi [2]

Nei capitoli successivi verranno enunciati nel dettagli le potenzialità e i limiti che ad oggi sono presenti nei software di sintesi del flusso OpenROAD

2.2.2 Floorplan e PDN (Power Delivery Network): TritonFP

Lo step successivo del flusso OpenROAD concerne la parte relativa al floorplan, definito come la rappresentazione sommaria del piazzamento provvisorio dei macro blocchi che compongono il design, e alla definizione della rete di distribuzione dell'alimentazione (PDN). Si veda come esempio la figura 2.4 che rappresenta il risultato finale dopo il

floorplanning e dopo la sistemazione della PDN di un elemento dell'architettura presa come modello, la NVDLA.

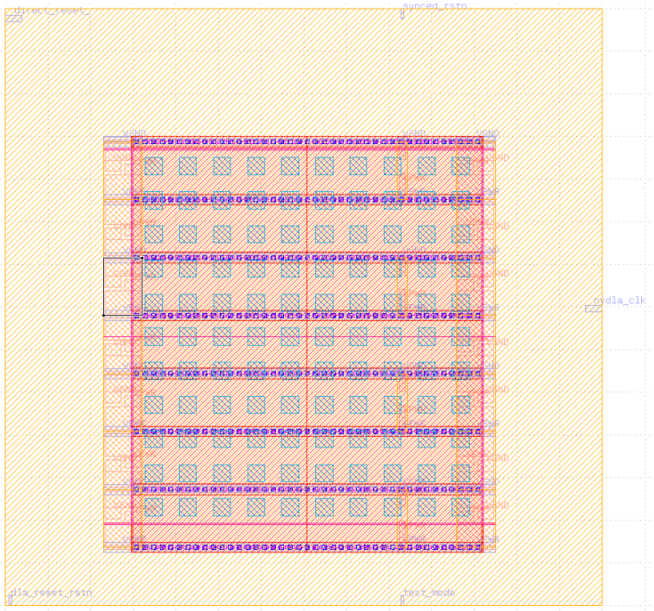


Figura 2.4: Floorplan e PDN del NDVLA_reset

«Il floorplanning nel flusso da RTL a GDSII inizia dagli output della sintesi logica e termina con un file in formato .DEF (Design Exchange Format), estensione open-source atta a rappresentare il layout fisico di un IC in un formato ASCII (American Standard Code for Information Interchange), che è adatto per il posizionamento globale delle celle standard, cioè con il piazzamento degli I/O (Input/Output) completato, delle macro, il layout della PDN (rete di Power e Ground) e il posizionamento delle welltap/endcap»[3]. La fase di floorplanning di OpenROAD è eseguita da TritonFP, che incorpora i diversi FOSS utilizzati per eseguire gli step di cui sopra:

- Resizer: tool di conversione Verilog-DEF;
- ioPlacer: tool di piazzamento dei pin I/O;
- RePlAce: tool di piazzamento globale;
- TritonMacroPlace: tool di piazzamento delle macro;
- PDN: tool di generazione del layout della rete di alimentazione;
- Tapcell: tool di inserimento di welltap/endcap.

Il software ioPlacer «sviluppato all'UFRGS (Università federale del Rio Grande do Sul) determina euristicamente le posizioni dei pin di I/O attraverso applicazioni dell'algoritmo di corrispondenza ungherese»[3], algoritmo il cui scopo è quello di risolvere il

problema delle assegnazioni, ovvero l'allocazione ottimale di diverse attività, utilizzando risorse minime. Ovviamente in questo contesto è utilizzato per trovare la migliore soluzione nella fase di decisione di posizionamento reciproco dei vari pin. «L'input consiste nel floorplan inizializzato in formato .DEF»[3], quindi l'output di Resizer, «e nella netlist post-sintesi, che può includere macro. Il codice ioPlacer accetta anche i vincoli sui livelli dei pin e i segmenti consentiti/non consentiti del confine della regione lungo i quali vengono posizionati gli I/O»[3].
Per quel che riguarda RePlAce: «è un placer open-source basato sull'elettrostatica»[3], ovvero che modella ogni oggetto come carica positiva e il costo di densità come energia potenziale del sistema elettrostatico, «che supporta circuiti di dimensioni miste (macro e celle)»[3].
TritonMacroPlace viene utilizzato «per semplificare le fasi successive di posizionamento e routing, ... divide la regione del layout in quattro quadranti e usa una versione modificata del software ParquetFP per circoscrivere e inserire le macro negli angoli del layout»[3].
Le posizioni delle macro generate forniscono un punto di partenza da cui vengono create più soluzioni di floorplan. Per ciascuna di essa, generate con macro fisse e PDN, RePlAce viene utilizzato nuovamente per determinare la migliore soluzione secondo un criterio che tiene conto della lunghezza totale delle interconnessioni.
Anche in questo caso, per le potenzialita e i limiti dei tool, si rimanda ai successivi capitoli.

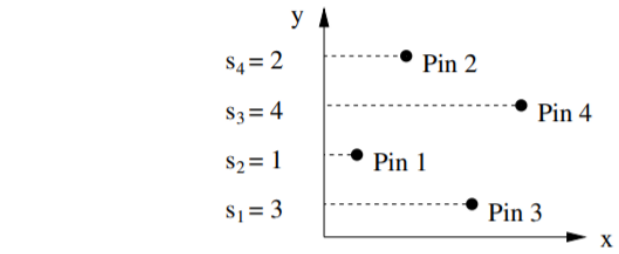
2.2.3 Placement

Il flusso OpenROAD sfrutta due software per eseguire la fase di piazzamento: RePlAce per il piazzamento globale ed OpenDP (Open-Source Detailed Placement Engine) per quello dettagliato.

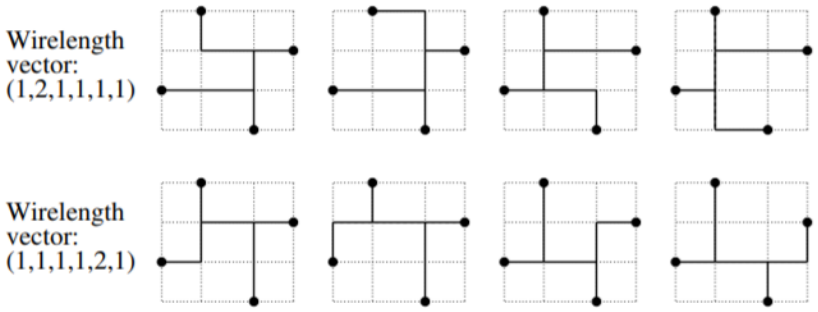
Global Placement: RePlAce

RePlAce è un FOSS ampiamente utilizzato nel flusso OpenROAD.
Viene adoperato durante la fase di floorplanning, come detto in precedenza, per il piazzamento a dimensione mista (celle e macro); lo si trova, ovviamente, sfruttato durante questa fase di piazzamento globale delle celle standard; ed infine è utilizzato durante lo step di CTS per eliminare le sovrapposizioni tra gate e macro (legalizzazione) dei buffer dell'albero di clock.
Come indicato nel paragrafo precedente, si tratta di un «placer analitico open-source con licenza BSD (Berkeley Source Distribution) basato su un'analogia con l'elettrostatica»[4].
Il tool è in grado di leggere i file nei diversi formati usati in ambito industriale, .LEF (Library Exchange Format), .DEF, Verilog, .SDC (Synopsys Design Constraint) e Liberty, ed incorpora diversi elementi atti ad ottenere il miglior risultato possibile di piazzamento in termini di stima delle lunghezze delle interconnessioni (utilizzando la tecnica FLUTE), stima dei parassiti e di analisi temporale statica (adoperando il tool open-source OpenSTA).
La tecnica denominata FLUTE si basa sull'idea che «l'insieme di tutte le reti di grado n-esimo»[5], dove per grado della rete si intende il numero di pin in essa compresi, «può essere suddiviso in $n!$ gruppi secondo le posizioni relative dei loro pin. Per ogni gruppo, la

lunghezza delle interconnesioni di tutte le possibili topologie di routing può essere scritta come un piccolo numero di combinazioni lineari di distanze tra i pin adiacenti. Chiamiamo ogni combinazione lineare un vettore di lunghezza di interconnessione potenzialmente ottimale (POWV - Potential Optimal Wirelength Vector). Memorizziamo i pochi POWV per ogni gruppo in una tabella. Per trovare la lunghezza ottimale abbiamo solo bisogno di calcolare le lunghezze dei fili corrispondenti ai POWV per il gruppo a cui la rete appartiene e poi riportare quella con la lunghezza minima del filo»[5]. Nelle figure 2.5(a) e 2.5(b) sono mostrate una sequenza verticale di una rete con 4 pin e le rappresentazioni dei POWV associati.



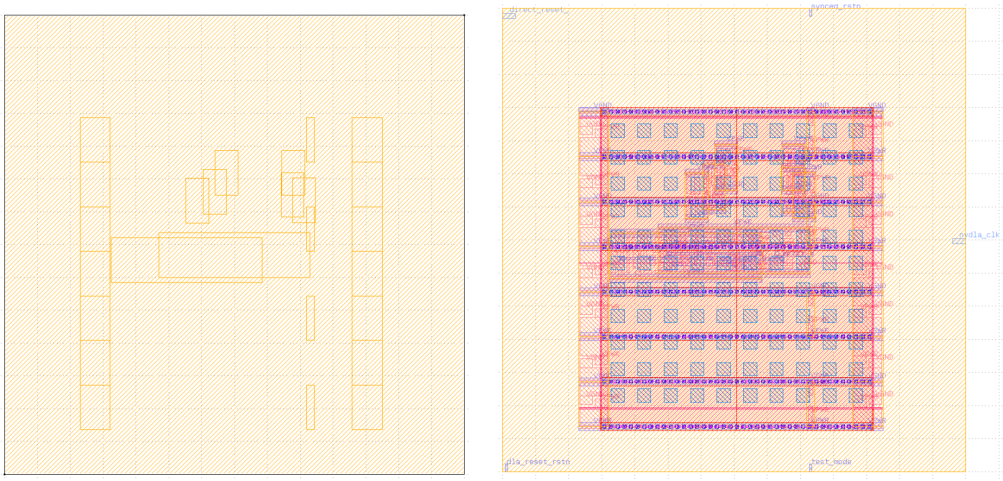
(a)



(b)

Figura 2.5: (a)Illustrazione di una sequenza verticale di una rete. (b) Potenziali soluzioni di routing ottimale per la reta in (a) [5]

Per quel che riguarda OpenSTA si rimanda ad un successivo paragrafo di questo capitolo.
Va notato che attualmente RePlAce non è in grado di modificare, in termini di buffer e dimensioni, la netlist generata dallo step di sintesi.
Ovviamente il passo di placement prevede una ridefinizione della PDN «basata su una migliore stima della distribuizione spaziale della corrente»[4].
In figura 2.6 si può osservare il risultato ottenuto dal piazzamento globale del solito elemento facente parte di NVDLA, NVDLA_reset. In figura 2.6(a) sono state evidenziate esclusivamente le celle piazzate mentre in 2.6(b) il risultato completo di PDN.

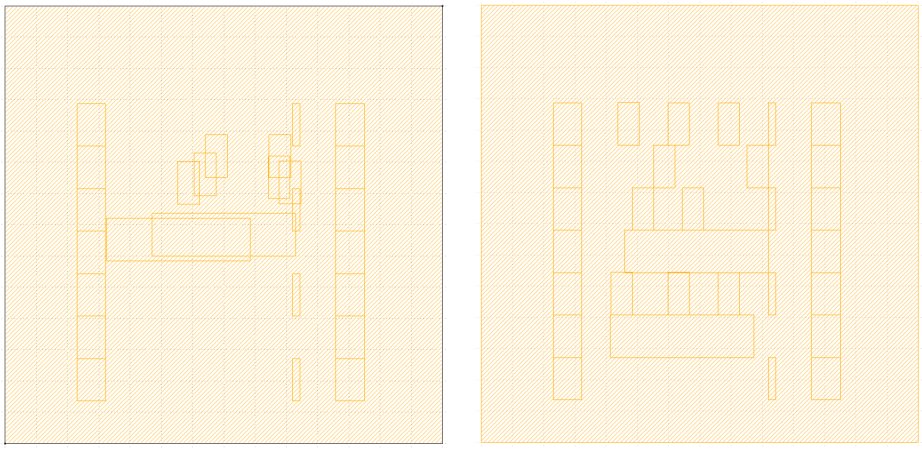


(a) (b)

Figura 2.6: (a)Risultato post global placement per NVDLA_reset, con celle standard evidenziate. (b) Rappresentazione completa post-placement

Detailed Placement: OpenDP

Nella versione di OpenROAD utilizzata durante questo lavoro di tesi la fase di piazzamento dettagliato è gestista dal software OpenDP.
Avendo precedentemente definito il concetto di "legalizzazione" del piazzamento, quindi la non sovrapposizione tra gate e macro, si può affermare che: «OpenDP esegue la legalizzazione basandosi sui vincoli relativi all'area in cui il design è inscritto. Esegue prima la pre-legalizzazione e la legalizzazione delle celle standard , poi migliora la qualità della soluzione usando il SA (simulated annealing) per ridurre lo spostamento dalle posizioni originali delle celle»[3].
In figura 2.7(b) si può notare il risultato dello step di piazzamento dettagliato del blocco analizzato precedentemente, NVDLA_reset, ed il confrontro con il risultato post-global placement, figura 2.7(a).



(a) (b)

Figura 2.7: (a)Risultato post global placement per NVDLA_reset, con celle standard evidenziate. (b) Risultato post detailed placement per NVDLA_reset, con celle standard evidenziate

2.2.4 CTS: Triton CTS

Il tool utilizzato per la sintesi dell'albero di clock è TritonCTS.
TritonCTS «esegue la CTS per una distribuzione del clock a bassa potenza, a basso skew e a bassa latenza, basato sul paradigma GH-Tree (H-Tree generalizzato) di Han»[4] ovvero una topologia di albero bilanciato che può avere una sequenza arbitraria di fattori di ramificazione. In figura 2.8 se ne mostra un esempio.

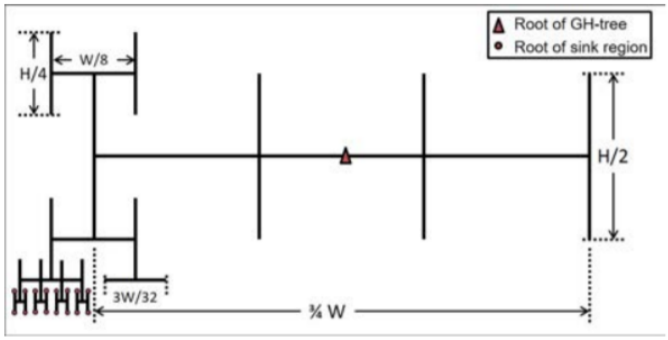
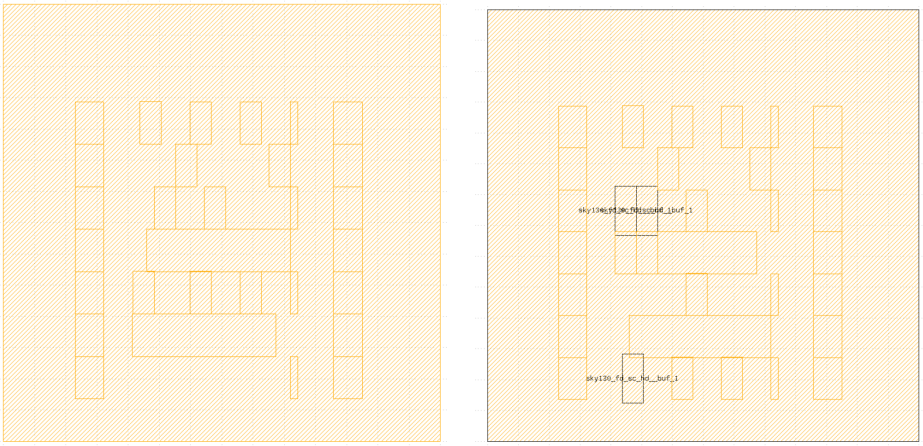


Figura 2.8: Esempio di GH-tree [4]

Elemento centrale del tool è un algoritmo dinamico in grado di stimare la soluzione di CTS a minor potenza, coerente con una latenza ed uno skew preimpostati. «La programmazione lineare è utilizzata per eseguire il clustering dei sink e il posizionamento dei buffer di clock»[1].
Peculiarità dell'uso di TritonCTS nel flusso OpenROAD è la possibilità di interazione tra

il tool di CTS con quello di placement (RePlAce) e quello di routing (TritonRoute). Di quest'ultimo si parlerà nel prossimo paragrafo.
Di fatto «il placer è usato per la legalizzazione dei buffer di clock inseriti. Il router mappa i pin dei sink sulle GCELL (Global Routing Cell) che dovrebbero essere usati per l'instradamento dell'albero di clock»[1]. In figura 2.9 si può vedere il risultato post-detail placement, (a), e post-CTS, (b), per il blocco NVDLA_reset. Nell'immagine 2.9(b) sono stati evidenziati i buffer inseriti da TritonCTS che delineano l'albero di clock.



(a) (b)

Figura 2.9: (a)Risultato post detailed placement per NVDLA_reset, con celle standard evidenziate. (b) Risultato post CTS per NVDLA_reset, con celle standard evidenziate

Per concludere l'analisi relativa al CTS si aggiunge che l'ultimo step prima della realizzazione delle interconnessioni consiste nell'inserimento delle celle FILLER, prive di funzione logica ma utilizzate per riempire gli spazi vuoti tra celle standard ed evitare DRV (Design Rule Violations) del tipo "NWell minimum spacing not met" e garantire la continuità delle N-WELL. In figura 2.10 viene mostrato il risultato dopo l'inserimento dei FILLER. Per una maggior chiarezza sono state evidenziate le celle con effettiva funzione logica.
Ovviamente l'inserimento di tali celle ha delle implicazioni a livello di potenza dissipata dal chip e di timing, ma di questi aspetti si rimanda al capitolo relativo all'analisi dei risultati di P&R

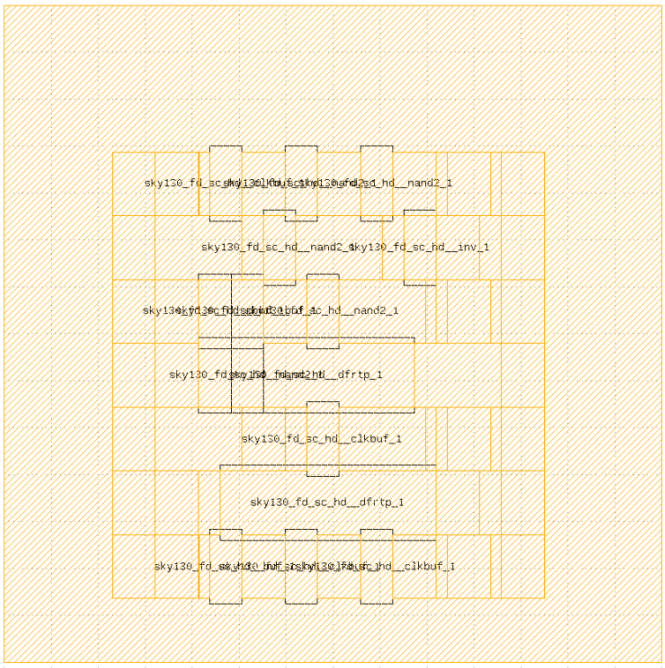


Figura 2.10: Risultato post-CTS e post inserimento FILLER cells per NVDLA_reset

2.2.5 Routing

In analogia con il passo relativo al piazzamento, il flusso OpenROAD sfrutta due tool separati per eseguire le fasi di routing globale e dettagliato.
La scissione dei tipi di routing, come per il placement, nasce come soluzione all'aumento della complessità dei design. Eseguire un'operazione generale per poi scendere sempre più nel dettaglio si è rivelata essere la soluzione migliore per ottenere i voluti risultati.
Il global routing è affidato al software FastRoute, mentre il detailed a TritonRoute.

Global Routing: FastRoute

«Il routing globale funziona su caselle astratte. Alloca la domanda di routing globalmente sull'area del circuito e guida il successivo routing dettagliato per terminare l'assegnazione delle piste e la creazione dei VIA»[6].
FastRoute è un FOSS ad opera dell'Università Statale dell'IOWA. Il software integra diverse tecniche: «la rapida costruzione dell'albero di Steiner tenendo conto dei VIA presenti e guidata della minimizzazione della congestione , il routing a 3 curve, l'aggiustamento della capacità virtuale, il maze routing multi-source multi-sink e l'assegnazione del livello a spirale»[6]. In figura 2.11 viene mostrato il flusso utilizzato di FastRoute per la generazione delle interconnessioni globali.

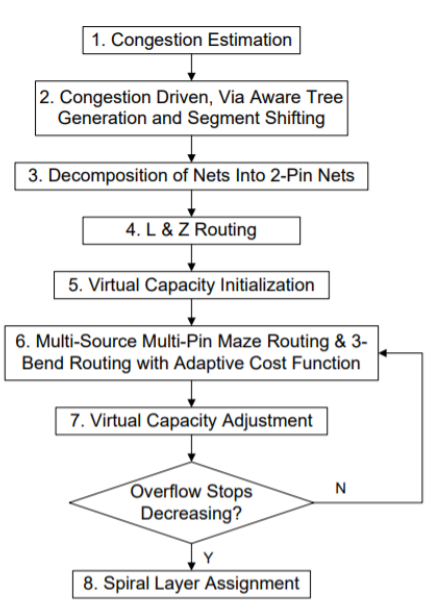


Figura 2.11: Flow FastRoute [6]

Per un'analisi dettagliata del tool si rimanda ad [6].
In questa sede si analizzeranno solo sommariamente le peculiarità del sofware, che consistono in:

- Una struttura accuratamente progettata per eseguire il routing globale in 3D in modo efficace ed efficiente;
- Una tecnica di generazione di alberi di Steiner guidata dalla minimizzazione della congestione e consapevole dei VIA per formare buone topologie di partenza per reti multi-pin;
- Una tecnica di spostamento dei segmenti per dirigere la domanda di routing dalla regione congestionata spostando alcuni bordi dell'albero senza aumentare la lunghezza del filo;
- Una tecnica di routing a 3 curve per esplorare rapidamente i percorsi di routing tra un pin sorgente e un pin sink con un equilibrio tra riduzione della congestione e controllo del numero di VIA;
- Una tecnica di maze routing multi-source multi-sink per ricollegare due sottoalberi in una rete multi-pin senza fissare i punti finali su entrambi i sotto-alberi;
- Una tecnica di capacità virtuale: un modo sistematico di guidare il routing a labirinto per evitare le regioni congestionate;
- Una nuova funzione di costo adattativa basata sulla funzione logistica per dirigere il routing a 3 curve e il maze routing per trovare percorsi meno percorsi congestionati;

- Una tecnica di assegnazione dei livelli a spirale per estendere una soluzione di routing 2D nella sua controparte 3D.

FastRoute genera come output un file nei formati industiali LEF/DEF ed in più un file denominato "route guide" «per ottenere un handoff (passaggio) ben definito tra il routing globale e dettagliato»[3].

Detail Routing: TritonRoute

«Dal file "route guide"»[7], di cui si riporta uno schema di esempio in figura 2.12, «generato come output finale del passo di routing globale, la fase iniziale di routing dettagliato dovrebbe generare una soluzione che rispetti il più possibile tale guida, minimizzando al contempo la lunghezza del filo, il numero di VIA e le varie DRV»[7].
TritonRoute è il tool utilizzato per lo step di routing dettagliato. Esso è composto da diversi blocchi costituitivi che vanno dall'analisi dei pin di accesso, all'assegnazione delle piste, alla ricerca e riparazione delle DRV ed un motore per il controllo delle regole di progetto (DRC - Design Rule Check).

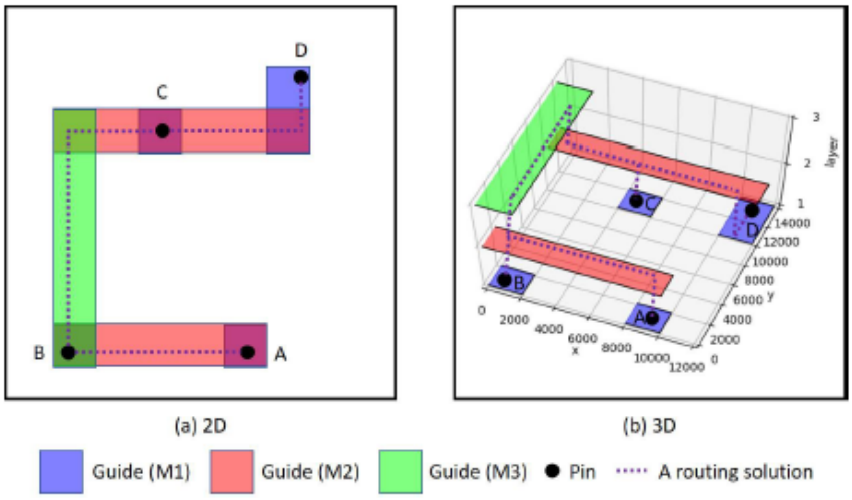


Figura 2.12: Un esempio di "route guide" per una rete a quattro pin: (a) vista dall'alto 2D e (b) vista 3D [7]

Come nel paragrafo relativo al routing globale, per una trattazione più dettagliata relativa al FOSS TritonRoute si rimanda all'articolo [7]. Se ne evidenziano qui le caratteristiche fondamentali:

- Un framework di routing parallelo basato sul MILP (Mixed Integer-Linear Programming) in grado di comprendere i vincoli di connettività (per esempio, circuiti aperti e cortocircuiti) e i vincoli delle regole di progettazione (cioè, tabelle di spazio, spazio di end-of-line (EOL), area minima e spazio di taglio);

- Un flusso complessivo che organizza il routing parallelo intra-layer all'interno di un quadro complessivo di routing sequenziale inter-layer. TritonRoute lavora in parallelo all'interno di ogni livello metallico, completa il routing su un layer e poi passa ad elaborare il superiore.

Il team dell'UC San Diego che ha sviluppato il software riporta in [7] i dati relativi ai risultati ottenuti da TritonRoute nel contesto dell'ISPD-2018 contest (International Symposium on Physical Design): «Abbiamo valutato il nostro router utilizzando la suite di benchmark ufficiale ISPD-2018 e lo script di valutazione e abbiamo mostrato che riduciamo le DRV fino al 93,58%, la metrica complessiva del concorso fino al 74% e in media il 50%, rispetto alla soluzione vincitrice del concorso per ogni testcase»[7].

2.2.6 Static Timing Analysis: OpenSTA

L'analisi statica dei tempi (STA) è un'attività chiave nel processo di progettazione di un IC.
É un metodo per convalidare le prestazioni di temporizzazione di un progetto controllando tutti i possibili percorsi per le violazioni dei tempi. L'analisi suddivide un design in percorsi di temporizzazione, calcola il ritardo di propagazione del segnale lungo ogni percorso e controlla le violazioni dei vincoli di temporizzazione all'interno del progetto e all'interfaccia di ingresso/uscita. Un esempio di percorsi di temporizzazione è mostrato in figura 2.13.

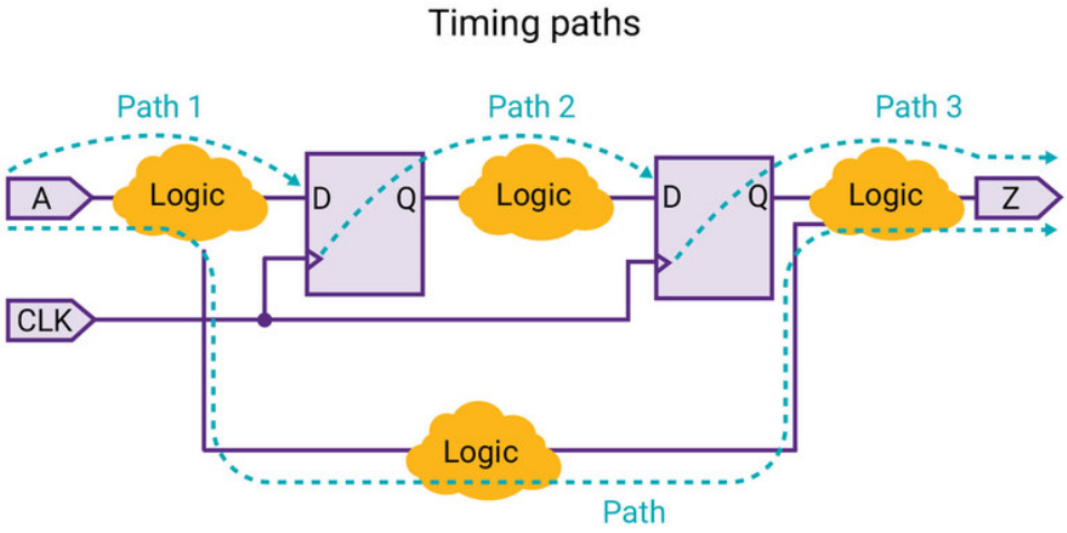


Figura 2.13: Un esempio di "timing paths"

Ogni "nuvola" logica rappresenta una rete di logica combinatoria ed un percorso di temporizzazione inizia (start point) dove i dati sono inviati da un fronte di clock o dove i dati devono essere disponibili in un momento specifico, passa attraverso una rete combinatoria e finisce (end point) dove i dati sono catturati da un fronte di clock o dove i dati

devono essere disponibili in un momento specifico.
Il tool che si occupa della STA nel flusso OpenROAD è OpenSTA, «una versione GPL3 (General Public Licence ver.3) open-source del timer commerciale Parallax»[1].
OpenSTA è un analizzatore di temporizzazione statica a livello di gate. Può essere usato per verificare il timing di un progetto usando formati di file standard:

- Netlist Verilog;
- Librerie Liberty;
- Vinvoli temporali SDC;
- Annotazioni di ritardo SDF (Standard Delay Format);
- Parassiti SPEF (Standard Parasitic Exchange Format).

OpenSTA usa un interprete di comandi TCL (Tool Command Language) per leggere il progetto, specificare i vincoli di temporizzazione e stampare i report di temporizzazione.

Capitolo 3

Architettura NVDLA: Acceleratore di Deep Learning

Nel contesto in cui questa tesi si inserisce, ovvero l'investigazione della possibilità di utilizzare un flusso di implementazione di IC completamente open-source, la scelta dell'architettura da analizzare per testare le capacità di questo progetto è ricaduta sull'acceleratore di Deep Learning open-source progettato da NVIDIA, NVDLA.
Si tratta appunto di un Neural Processing Unit (NPU), ovvero di un processore dedicato per l'accelerazione hardware della fase di inferenza di un algoritmo di Deep Learning. Si ricorda brevemente cosa si intende per Deep Learning. Si tratta di una branca della scienza riguardante gli algoritmi di intelligenza artificiale (AI Artificial Intelligence). Tale scienza si divide in Machine Learning, ovvero la capacità di un sistema di imparare dai dati, ed in Deep Learning, appunto, che implica la sintesi di reti neurali artificiali (ANN Artificial Neural Network), in grado di simulare il cervello umano e di evolversi secondo un andamento "naturale e non imposto".
L'inferenza è una delle due fasi previste dal Deep Learning. La prima consiste nell'addestramento, in cui vengono forniti alla rete grandi quantità di dati, descritti da particolari etichette e caratterizzati nel modo più esaustivo possibile. A questa fase segue appunto l'inferenza, in cui la rete, sulla base dei dati forniti precedentemente, è in grado di determinare autonomamente le etichette e le caratteristiche principali di nuovi dati.

3.1 CNN: Reti Neurali Convoluzionali

Una rete neurale convoluzionale (CNN Convutional Neural Network) è uno specifico algoritmo del deep learning costituito da "neuroni" con pesi e bias di apprendimento, utilizzato in particolare per il rilevamento di immagini.
Le immagini possono essere caratterizzate da una dimensione molto alta, quindi applicare una rete neurale completamente connessa può essere molto difficile dal punto di vista computazionale. L'idea è quella di "dividere" l'immagine in diverse parti ed estrarne le caratteristiche locali, per poi unificare queste singole porzioni alla fine del processo. L'algoritmo delle CNN è caratterizzato principalmente da tre livelli: convoluzione, attivazione e pooling.

Ogni neurone esegue un prodotto scalare del suo input e dei suoi pesi seguito da un'attivazione non lineare.

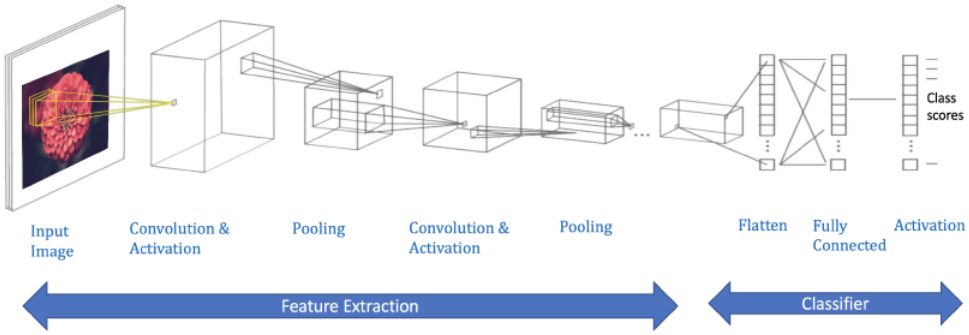


Figura 3.1: Esempio degli strati di una CNN[8]

NVDLA coinvolge diversi blocchi funzionali che eseguono queste 3 operazioni.

3.2 NVDLA: NVIDIA Deep Learning Accelerator

«Il progetto NVIDIA® Deep Learning Accelerator (NVDLA) promuove un'architettura standardizzata e open-source per affrontare le richieste computazionali dell'inferenza. L'architettura NVDLA è sia scalabile che altamente configurabile; il design modulare mantiene la flessibilità e semplifica l'integrazione. La standardizzazione dell'accelerazione del Deep Learning promuove l'interoperabilità con la maggior parte delle moderne reti di Deep Learning e contribuisce a una crescita unificata del machine learning su larga scala»[8]. Si tratta di un'architettura profondamente scalabile e modulare in grado quindi di operare a diversi livelli di complessità, in base all'applicazione d'interesse.
«NVDLA viene fornito come un insieme di modelli IP-core basati su standard industriali open-source: i modelli Verilog di sintesi e simulazione sono in forma RTL, e il modello di simulazione TLM (Transaction-Level Model) SystemC può essere utilizzato per lo sviluppo del software, l'integrazione del sistema e il test. L'ecosistema software di NVDLA include uno stack software on-device (parte della release open source), un'infrastruttura di formazione completa per costruire nuovi modelli che incorporano il Deep Learning, e parser che convertono i modelli esistenti in una forma utilizzabile dal software on-device.»[8]
Come detto in precedenza, l'architettura ha una struttura modulare, composta dai seguenti elementi:

- Convolution Core - motore di convoluzione ottimizzato ad alte prestazioni;
- Single Data Processor - motore di ricerca a punto singolo per le funzioni di attivazione;
- Planar Data Processor - motore di mediazione planare per il pooling;

- Channel Data Processor - motore di mediazione multicanale per funzioni di normalizzazione avanzate;
- Dedicated Memory and Data Reshape Engines - accelerazione della trasformazione da memoria a memoria per la rimodellazione dei tensori e le operazioni di copia.

ed in funzione del campo di lavoro alcuni di questi elementi possono essere presenti nella specifica implementazione o meno, senza che le effettive prestazioni dell'accelerazione ne risentano. In figura 3.2 è mostrata uno schema esplicativo dell'architettura.

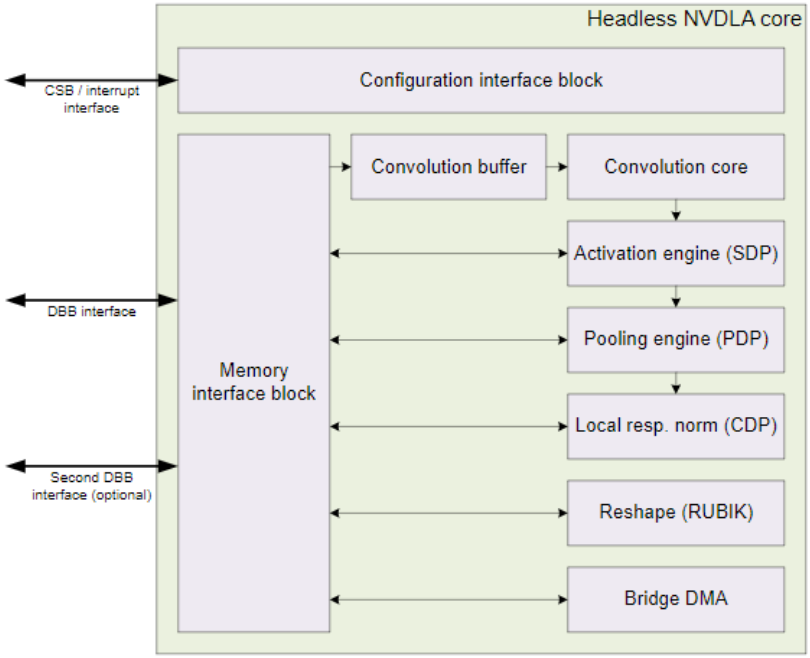


Figura 3.2: Architettura NVDLA[8]

3.2.1 Modelli verilog

«Il modello Verilog fornisce un modello di sintesi e simulazione in forma RTL. Ha quattro interfacce funzionali: un'interfaccia host slave, una linea di interrupt, e due interfacce master per l'accesso alla memoria interna ed esterna. Le interfacce host e di memoria sono molto semplici, ma richiedono adattatori di bus esterni per connettersi a un progetto SoC esistente; per comodità, adattatori campione per AXI4 e TileLink sono inclusi come parte della release open source di NVDLA. La release open source di NVDLA contiene script di sintesi di esempio. Per facilitare la progettazione fisica su sistemi più complessi o istanze più grandi di NVDLA, il progetto è diviso in partizioni che possono essere gestite indipendentemente nel flusso backend del SoC. Le interfacce tra le partizioni possono essere reimpostate come necessario per soddisfare i requisiti di routing».[8] In Appendice

viene riportato lo script di sintesi utilizzato durante questa implementazione.
Le partizioni che compongono NVDLA sono:

- Partizione_a: Questo modulo accumula le somme parziali dalle matrici MAC e stima i risultati prima di inviarli alla fase successiva delle attivazioni;
- Partizione_c: Questa sezione gestisce varie operazioni del kernel di convoluzione come CDMA, CBUF e CSC.
  - CDMA - Convolution DMA recupera i dati dalla SRAM/DRAM e li memorizza in un buffer di convoluzione. Comprende due porte di lettura, vale a dire la porta di lettura del peso e la porta di lettura dei dati che si collegano all'interfaccia AXI per ottenere i dati del peso/caratteristica;
  - CBUF - Il buffer di convoluzione è lo stadio successivo della pipeline. È una cache SRAM di 512KB che memorizza i dati di input e i pesi;
  - CSC - Il Convolution Sequence Controller carica i dati memorizzati dal buffer alle rispettive unità MAC in modo appropriato. Così, controllando la sequenza di calcolo nella pipeline di convoluzione.
- Partizione_m: Questa partizione esegue i calcoli di moltiplicazione e addizione. Questo modulo CMAC (Convolution MAC) comprende 16 celle MAC. Ogni di queste celle include 64 moltiplicatori a 16 bit e 72 adders;
- Partizione_p: Questa sezione esegue varie operazioni lineari e non lineari
- Partition_o : Questa sezione controlla la comunicazione tra gli elementi di elaborazione con i controllori esterni e le unità di memoria.
  - CSB - Questo modulo legge e scrive i registri di configurazione di ogni livello nel core NVDLA. Questo trasferisce i dati dal processore di gestione esterno attraverso l'interfaccia APB;
  - CFGROM - Questo mantiene i parametri configurabili del core per la sua rispettiva definizione delle specifiche;
  - MCIF - Questa interfaccia comunica con tutte le sottounità che accedono alla DDR esterna. Questo bus dati utilizza un protocollo AXI;
  - PDP e CDP - Queste unità eseguono il pooling e la normalizzazione della risposta locale rispettivamente;
  - GLB - Controllano i segnali di interrupt in uscita di tutti i sub core di NVDLA.

Come accennato, durante il lavoro di implementazione dell'architettura ci siamo limitati allo studio della singola partizione_a.
In figura 3.3 si mostra la struttura partizionata di NVDLA

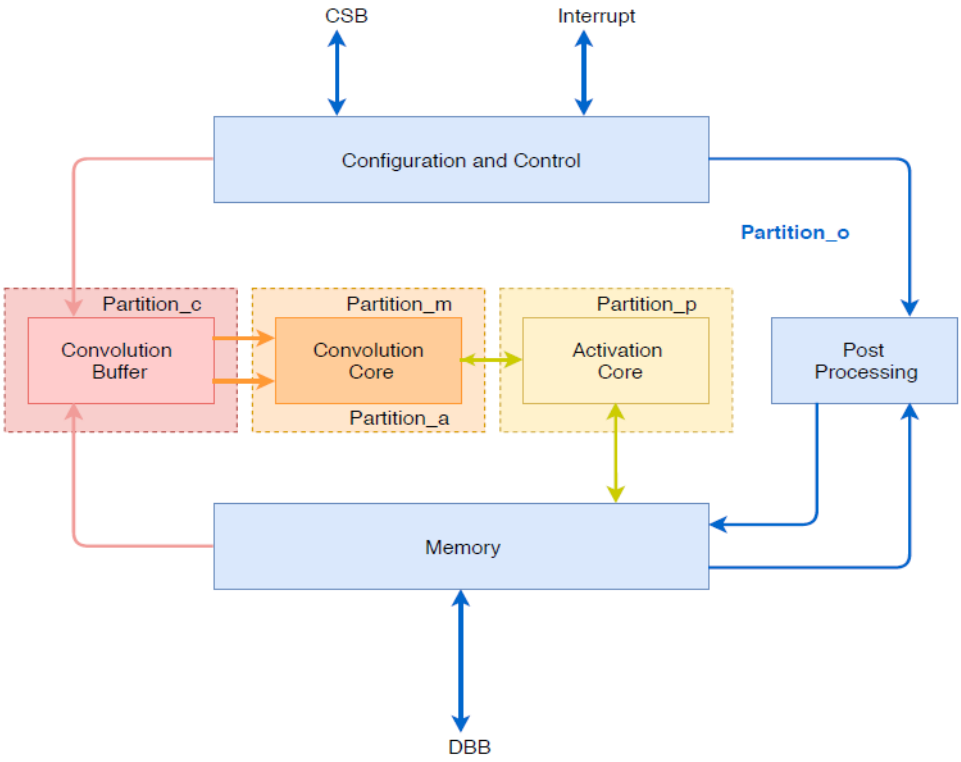


Figura 3.3: Partizioni NVDLA [9]

3.2.2 NVDLA_partition_a

Come detto in precedenza la partizione_a dell'acceleratore è quella che si occupa dell'accumulo delle somme parziali della matrici di MAC.
Il cuore di questa partizione è il CACC (Convolution Accumulator), l'ultimo stadio della pipeline di convoluzione, una delle pipeline interne alla logica del NVDLA, come mostrato in figura 3.4.

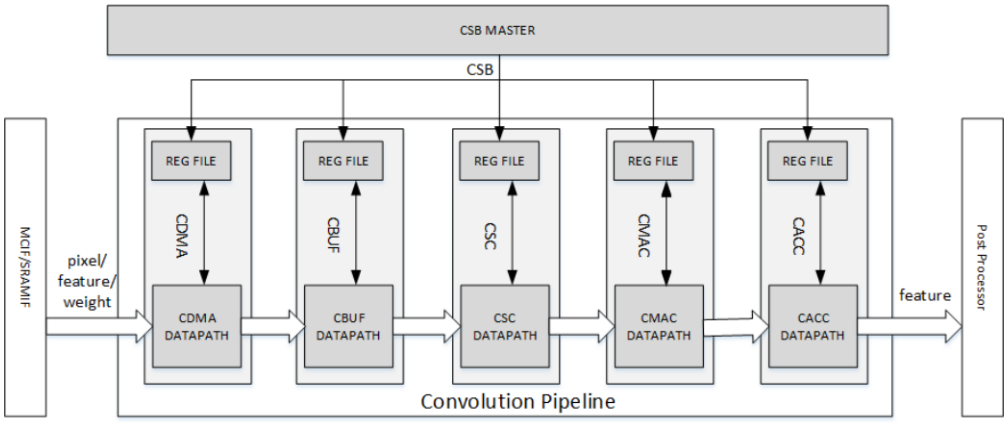


Figura 3.4: Pipeline di convoluzione[8]

Tale elemento «È usato per accumulare le somme parziali ricevute dal Convolution MAC, e arrotondare/saturare il risultato prima di inviarlo alla SDP (Single Data Processor)»[8]. La struttura interna è mostrata in figura 3.5

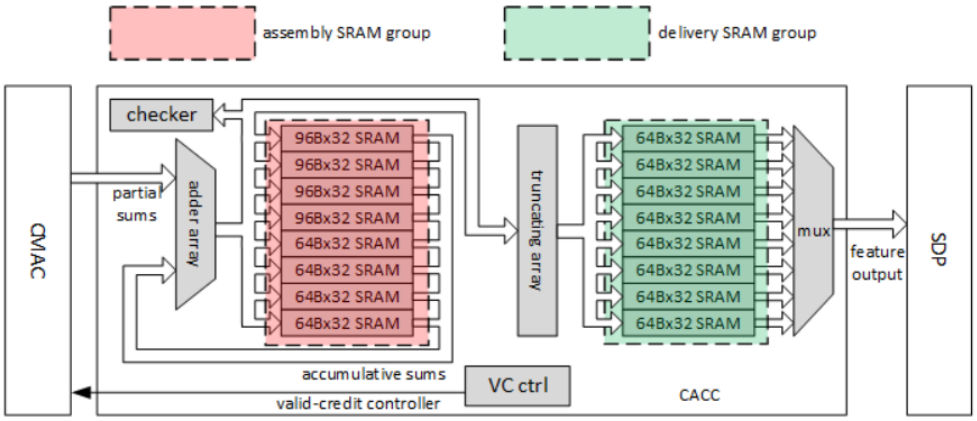


Figura 3.5: Struttura interna CACC[8]

«Il flusso di lavoro dell'accumulatore è il seguente:

- Prefetch delle somme cumulative dal gruppo assembly SRAM;
- Quando arrivano le somme parziali, le invia all'array di adder insieme alle somme cumulative. Se le somme parziali provengono dalla prima operazione di stripe, le somme cumulative dovrebbero essere 0;
- Raccoglie nuove somme cumulative dal lato di uscita dell'array di adder;
- Memorizza nel gruppo di assembly SRAM;

- Ripete i passi 1-3 in termini di operazione di stripe fino a quando un'operazione di canale è terminata;
- Se viene terminata un'operazione di canale, l'uscita degli adders è arrotondata e saturata;
- Raccoglie i risultati del passo precedente e li memorizza nel gruppo di delivery SRAM;
- Carica i risultati dal gruppo delivery buffer e li invia al SDP.»[8]

Come si evince dall'immagine 3.5 e come riportato nel flusso soprastante, la partizione è dotata di due gruppi di SRAM, le assembly e le delivery SRAM. Si è gia accennato in introduzione come il codice sorgente dell'architettura sia sprovvisto di memorie sintetizzabili. Per risolvere questo problema abbiamo fornito al sintetizzatore dei modelli di SRAM sintetizzabili. Di questo aspetto si tratterà maggiormente nel capitolo relativo alla sintesi. In questa parte si accenna brevemente ad un framework utilizzabile per la compilazione di memorie: OpenRAM.

OpenRAM

«OpenRAM è un framework Python open-source per creare layout, netlist, modelli di timing e di potenza, modelli di posizionamento e routing necessari per utilizzare le SRAM nella progettazione ASIC. OpenRAM supporta l'integrazione in flussi sia commerciali che open-source con tecnologie sia predittive che fabbricabili»[10].

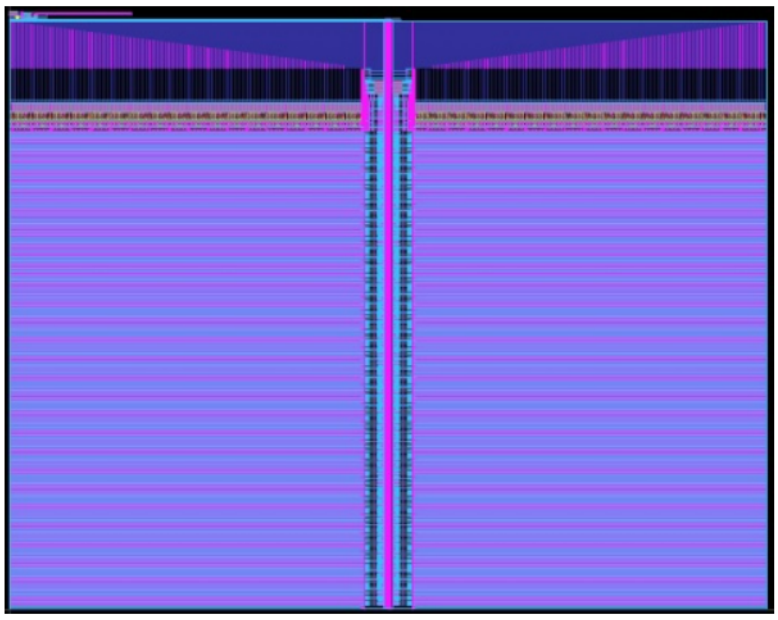


Figura 3.6: Layout di una SRAM a 2 banchi, da 16kbyte implementata su OpenRAM [10]

Il progetto OpenRAM nasce dalla considerazione che «La maggior parte delle metodologie accademiche di progettazione ICs sono limitate dalla disponibilità di memorie. Molti PDK che utilizzano standard cell sono disponibili dalle fonderie e dai fornitori, ma spesso non vengono forniti con array di memoria o compilatori di memoria. Se un compilatore di memoria è disponibile gratuitamente, spesso supporta solo una tecnologia di processo generica non fabbricabile.A causa delle restrizioni di finanziamento accademico, le soluzioni dell'industria commerciale spesso non sono fattibili per i ricercatori... Queste restrizioni e i problemi di licenza rendono impossibile il confronto e la sperimentazione con le memorie del mondo reale»[11].
In figura 3.7 è mostrata la divisione tra front-end e back-end del framework.

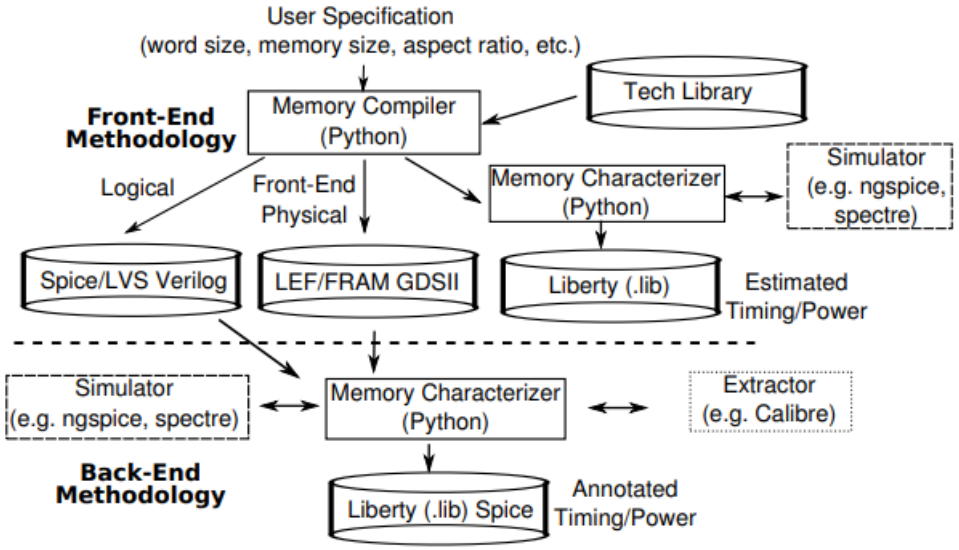


Figura 3.7: Organizzazione framework OpenRAM [11]

«Il frontend ha il compilatore e il caratterizzatore. Il compilatore genera modelli SPICE e i relativi layout GDSII basati sugli input dell'utente. Il caratterizzatore chiama un simulatore SPICE per produrre risultati di timing e potenza. Il back-end utilizza una netlist spice estratta dal layout GDSII per generare modelli di tempistica e potenza annotati»[11].
In conlcusione si analizza la struttura interna del singolo banco RAM statico sintetizzato tramite OpenRAM. È composta da 8 macro blocchi:

- Bit-cell Array;
- Address Decoder;
- Word-Line Driver;
- Column Multiplexer;

- Bit-line Precharge;
- Sense Amplifier;
- Write Driver;
- Control Logic.

La figura 3.8 ne mostra lo schema.

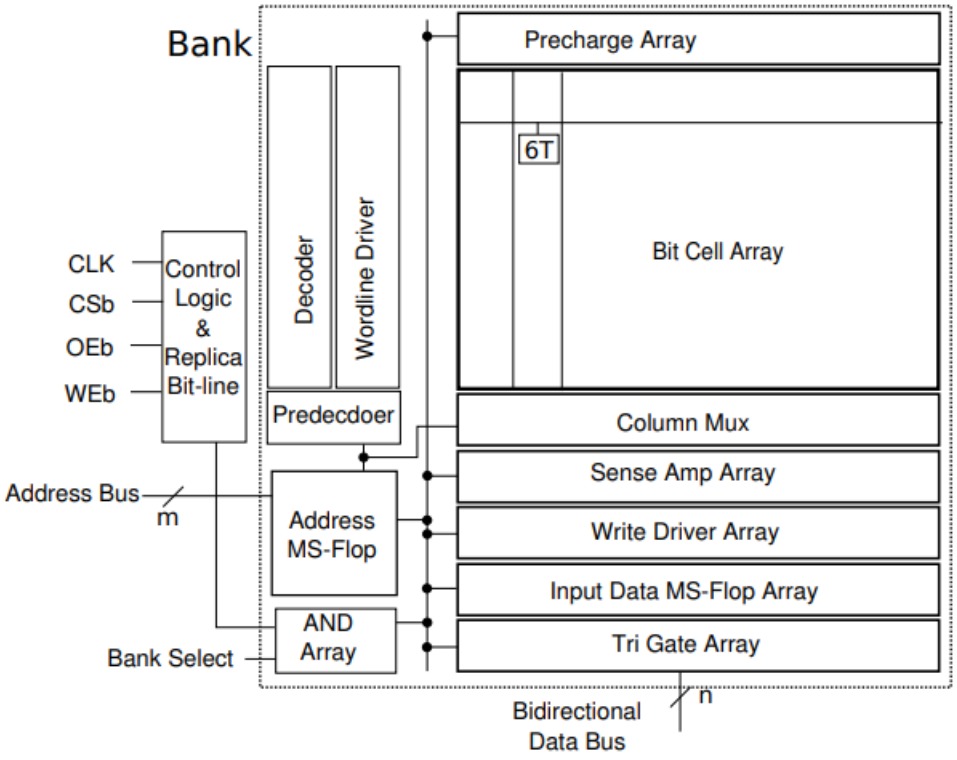


Figura 3.8: Struttura interna banco RAM su OpenRAM [11]

Capitolo 4

Sintesi

In questo capitolo verrano mostrati ed analizzati i risultati di sintesi logica di entrambi gli ambienti di lavoro, quelli relativi al FOSS Yosys/ABC, facente parte del flusso OpenROAD, e quelli ottenuti utilizzando il software proprietario di Cadence, Synopsys Design Compiler. Come detto in precedenza si osserveranno quelle che sono le prestazioni dei due framework in termini di timing (setup e hold time slack), potenza dissipata ed area occupata dalle diverse netlist.
Si è scelto di concentrare l'attenzione su una singola partizione dell'acceleratore NVDLA, NVDLA_partition_a. Questo per due ragioni:

- La semplificazione del lavoro di analisi;
- Difficoltà del progetto OpenROAD nel gestire strutture di dimesioni e complessità elevate.

Dal momento che lo scopo di questa tesi è quello di testare effettiavemente le capacità e le prestazioni del flusso OpenROAD nel contesto di implementazione di IC, e di effettuare una comparazione di tali prestazioni con quelle ottenibili utilizzando software con licenze proprietarie, si è deciso di effettuare un confrontro dei risultati di sintesi ottenuti nei seguenti tre contesti:

- Analisi post-sintesi di netlist generate da Yosys/ABC nell'ambiente OpenROAD;
- Analisi post-sintesi di netlist generate da Synopsys Design Complier nell'ambiente Synopsys;
- Analisi post-sintesi di netlist generate da Yosys/ABC nell'ambiente Synopsys;

Il primo e l'ultimo scenario danno anche indicazione di come i due ambienti, lavorando con le stesse netlist, quindi con le stesse porte logiche mappate, effettuino la stima di timing, potenza ed area.
L'analisi è stata svolta in modo gerarchico, andando quindi a lavorare con i singoli elementi che compongono l'intera partizione, si veda figura 5.1. Ovviamente verranno mostrati anche i risultati relativi all'intera NVDLA_partiton_a.

```
Top module:  \NV_NVDLA_partition_a
Used module:     \NV_NVDLA_RT_cmac_b2cacc
Used module:     \NV_NVDLA_cacc
Used module:         \NV_NVDLA_CACC_slcg
Used module:             \NV_CLK_gate_power
Used module:         \NV_NVDLA_CACC_delivery_buffer
Used module:         \NV_NVDLA_CACC_delivery_ctrl
Used module:         \NV_NVDLA_CACC_calculator
Used module:             \NV_NVDLA_CACC_CALC_fp_48b
Used module:             \NV_NVDLA_CACC_CALC_int8
Used module:             \NV_NVDLA_CACC_CALC_int16
Used module:         \NV_NVDLA_CACC_assembly_buffer
Used module:         \NV_NVDLA_CACC_assembly_ctrl
Used module:         \NV_NVDLA_CACC_regfile
Used module:             \NV_NVDLA_CACC_dual_reg
Used module:             \NV_NVDLA_CACC_single_reg
Used module:     \NV_NVDLA_sync3d_s
Used module:         \sync3d_s_ppp
Used module:             \p_SSYNC3DO_S_PPP
Used module:         \MUX2HDD2
Used module:         \NV_BLKBOX_SRC0
Used module:     \NV_NVDLA_sync3d
Used module:         \sync3d
Used module:             \p_SSYNC3DO
Used module:     \NV_NVDLA_reset
Used module:         \sync_reset
Used module:             \MUX2D4
Used module:             \p_SSYNC2DO_C_PP
Used module:             \OR2D1
```

Figura 4.1: Gerarachia NVDLA_partition_a

Verrano esaminate le prestazioni di entrambi gli ambiente per due frequenze di clock differenti: $2MHz$ e $200MHz$.
Nella trattazione che seguirà, per alcuni elementi che compongono la partizione verranno riportati due risultati per ogni metrica di confronto. Questa impostazione nasce dal fatto che nel codice verilog che descrive tali elementi vi è istanziato un componente, nel dettaglio NV_NVDLA_CACC_CALC_fp_48b, il quale utilizza un blocco facente parte della libreria proprietaria DesignWare di Synopsys Design Compiler, DW_lsd. Si tratta di un rilevatore di segno.
Il codice sorgente è provvisto di un'alternativa sviluppata da NVIDIA, NV_DW_lsd1 ed NV_DW_lsd2, per contesti di applicazione dell'acceleratore in cui tale libreria non è presente. Il flusso OpenROAD ne è un esempio.
Per ottenere un confronto il più esaustivo possibile si è, quindi, deciso di effettuare due sintesi su Synopsys Design Compiler, una che adopera la libreria DesignWare e l'altra in cui si sono utilizzate delle versioni lievemente modificate dei blocchi NV_DW_lsd1 ed NV_DW_lsd2.
Ultima considerazione da fare prima di procedere con l'analisi dei risultati di sintesi riguarda le RAM. Come accennato nel capitolo relativo all'architettura di riferimento, il codice sorgente è sprovvisto di modelli di memorie sintetizzabili. In mancanza di un compilatore di RAM, OpenRAM, sono stati implementati dei codici sintetizzabili di RAM in verilog, riportati in appendice B. Data la semplicità di tali codici, si sono

modificati entrambi gli ambienti, quello OpenROAD e quello Synopsys, per far trattare dai due sintetizzatori tali blocchi come black box. Essendo questi elementi analizzati esclusivamente per le loro interfacce input/output e non per la loro descrizione interna le stime di timing, potenza e area non ne tengono conto. Ne risulta che i dati riportati sono incompleti, ma nonostante ciò rapresentano, con una buona accuratezza, le prestazioni ottenibili dall'acceleratore.
In questo capitolo non verranno riportate le potenzialità e i limiti della sintesi effettuata in ambiente OpenROAD dal momento che questi aspetti verranno discussi in un capitolo successivo. Si accenna esclusivamente all'impossibilità da parte di Yosys/ABC di fornire dei report post-sintesi di potenza e timing. Le uniche informazioni riguardo al design che la sintesi open-source è in grado di riportare sono il numero di celle utilizzate e l'area del progetto.
I risultati relativi al flusso OpenROAD riportati nei prossimi paragrafi sono stati ottenuti utilizzando il tool di floorplanning, TritonFP, proprio per tale problematica.

4.1 Analisi temporale

L'analisi temporale risulta fondamentale durante un lavoro di progettazione per poter avere un'indicazione teorica della massima frequenza alla quale un sistema può lavorare senza incorrere in violazioni di setup e hold time. Di questi due concetti se ne ricordano le definizioni:

- Setup time: quantità di tempo necessaria affinché un segnale in ingresso ad un Flip-Flop sia stabile prima di un fronte di clock;
- Hold time: quantità minima di tempo necessaria affinché l'ingresso di un Flip-Flop sia stabile dopo un fronte di clock.

Come accennato in precedenza, sono state effetuate due sintesi. La prima ad una frequenza di clock estremamente rilassata, $2MHz$ (500ns), scelta semplicemente per testare le capacità del software di sintesi open-source. Ovviamente in tale contesto tutti i vincoli temporali sono stati rispettati e si ottengono esclusivamente slack positivi, dove per slack si intende la differenza tra i tempi di arrivo desiderati e il tempo di arrivo effettivo di un segnale. Uno slack positivo indica che il progetto è in grado di lavorare a quella frequenza operativa. Dalla quantità di slack positivo si può provare a ridurre il periodo di clock, tenendo sempre presente il trade-off tra frequenza di lavoro e potenza dissipata, la quale aumenta al crescere della frequenza.
Nelle tabella 5.1, 5.2 e 5.3 sono riportati i valori dei CPS (Critical Path Slack), quindi i percorsi tra un flip-flop ed un altro che presentano il minor valore di slack, i WNS (Worst Negative Slack), che coincide con il dato precedente nel caso in cui vi sia una violazione di setup time, ed il TNS (Total Negative Slack), che corrisponde alla somma di tutti gli slack negativi dei vari percorsi interni al design, nei tre scenari descritti nel paragrafo introduttivo alla sintesi:

- Analisi post-sintesi di netlist generate da Yosys/ABC nell'ambiente OpenROAD, tabella 5.1;

- Analisi post-sintesi di netlist generate da Synopsys Design Compiler nell'ambiente Synopsys, tabella 5.2;

- Analisi post-sintesi di netlist generate da Yosys/ABC nell'ambiente Synopsys, tabella 5.3;

nella condizione di frequenza operativa di $2MHz$.Sono stati indicati tali valori per i diversi blocchi che compongono la NVDLA_partition_a.

	CPS[ns]	WNS[ns]	TNS[ns]
NV_NVDLA_reset	499.54	0	0
NV_NVDLA_sync3d	499.61	0	0
NV_NVDLA_sync3d_s	499.43	0	0
NV_NVDLA_RT_cmac_b2cacc	498.84	0	0
NV_NVDLA_CACC_regfile	498.06	0	0
NV_NVDLA_CACC_assembly_ctrl	496.30	0	0
NV_NVDLA_CACC_assembly_buffer	249.26	0	0
NV_NVDLA_CACC_delivery_ctrl	497.56	0	0
NV_NVDLA_CACC_delivery_buffer	247.81	0	0
NV_NVDLA_CACC_calculator	494.74	0	0
NV_NVDLA_cacc	247.62	0	0
NV_NVDLA_partition_a	248.08	0	0

Tabella 4.1: Analisi temporale post-sintesi di netlist generate da Yosys/ABC nell'ambiente OpenROAD ad una frequenza operativa di $2MHz$

	CPS[ns]	WNS[ns]	TNS[ns]
NV_NVDLA_reset	499.5429	0	0
NV_NVDLA_sync3d	499.6135	0	0
NV_NVDLA_sync3d_s	499.4547	0	0
NV_NVDLA_RT_cmac_b2cacc	497.2633	0	0
NV_NVDLA_CACC_regfile	495.8562	0	0
NV_NVDLA_CACC_assembly_ctrl	496.7398	0	0
NV_NVDLA_CACC_assembly_buffer	246.5212	0	0
NV_NVDLA_CACC_delivery_ctrl	491.8344	0	0
NV_NVDLA_CACC_delivery_buffer	246.8283	0	0
	DW_lsd | NV_DW_lsd		
NV_NVDLA_CACC_calculator	488.2186 | 488.2186	0	0
NV_NVDLA_cacc	246.1487 | 246.1848	0	0
NV_NVDLA_partition_a	246.1848 | 246.1487	0	0

Tabella 4.2: Analisi post-sintesi di netlist generate da Synopsys Design Complier nell'ambiente Synopsys ad una frequenza operativa di $2MHz$

	CPS[ns]	WNS[ns]	TNS[ns]
NV_NVDLA_reset	499.5429	0	0
NV_NVDLA_sync3d	499.6135	0	0
NV_NVDLA_sync3d_s	499.4409	0	0
NV_NVDLA_RT_cmac_b2cacc	498.8821	0	0
NV_NVDLA_CACC_regfile	498.1731	0	0
NV_NVDLA_CACC_assembly_ctrl	496.4798	0	0
NV_NVDLA_CACC_assembly_buffer	249.2694	0	0
NV_NVDLA_CACC_delivery_ctrl	497.7071	0	0
NV_NVDLA_CACC_delivery_buffer	248.8371	0	0
NV_NVDLA_CACC_calculator	495.1176	0	0
NV_NVDLA_cacc	247.7408	0	0
NV_NVDLA_partition_a	248.1938	0	0

Tabella 4.3: Analisi post-sintesi di netlist generate da Yosys/ABC nell'ambiente Synopsys ad una frequenza operativa di $2MHz$

Come si può vedere i valori di CPS risultano elevati e non si hanno violazioni di setup time, come i valori nulli di WNS e TNS indicano.
Risultato diverso, invece, si ottiene riducendo il periodo di clock a 5ns, frequenza operativa di $200MHz$. Le tabelle 5.4, 5.5, 5.6 riportano i valori dell'analisi temporale nei soliti tre scenari.

	CPS[ns]	WNS[ns]	TNS[ns]
NV_NVDLA_reset	4.54	0	0
NV_NVDLA_sync3d	4.61	0	0
NV_NVDLA_sync3d_s	4.43	0	0
NV_NVDLA_RT_cmac_b2cacc	3.84	0	0
NV_NVDLA_CACC_regfile	3.06	0	0
NV_NVDLA_CACC_assembly_ctrl	1.30	0	0
NV_NVDLA_CACC_assembly_buffer	1.76	0	0
NV_NVDLA_CACC_delivery_ctrl	2.56	0	0
NV_NVDLA_CACC_delivery_buffer	1.26	0	0
NV_NVDLA_CACC_calculator	-0.26	-0.26	-5.65
NV_NVDLA_cacc	-0.79	-0.79	-3606.50
NV_NVDLA_partition_a	-0.57	-0.57	-1913.01

Tabella 4.4: Analisi temporale post-sintesi di netlist generate da Yosys/ABC nell'ambiente OpenROAD ad una frequenza operativa di $200MHz$

	CPS[ns]	WNS[ns]	TNS[ns]
NV_NVDLA_reset	4.5429	0	0
NV_NVDLA_sync3d	4.6135	0	0
NV_NVDLA_sync3d_s	4.4546	0	0
NV_NVDLA_RT_cmac_b2cacc	2.4764	0	0
NV_NVDLA_CACC_regfile	0.8562	0	0
NV_NVDLA_CACC_assembly_ctrl	2.0364	0	0
NV_NVDLA_CACC_assembly_buffer	0.8119	0	0
NV_NVDLA_CACC_delivery_ctrl	0.0035	0	0
NV_NVDLA_CACC_delivery_buffer	0.0885	0	0
	DW_lsd | NV_DW_lsd		
NV_NVDLA_CACC_calculator	0.0005 | 0.0013	0	0
NV_NVDLA_cacc	-0.0172 | 0	-0.0172 | 0	-13.0152 | 0
NV_NVDLA_partition_a	-0.2608 | 0	-0.2608 | 0	-832.8659 | 0

Tabella 4.5: Analisi post-sintesi di netlist generate da Synopsys Design Complier nell'ambiente Synopsys ad una frequenza operativa di $200MHz$

	CPS[ns]	WNS[ns]	TNS[ns]
NV_NVDLA_reset	4.5429	0	0
NV_NVDLA_sync3d	4.6135	0	0
NV_NVDLA_sync3d_s	4.4409	0	0
NV_NVDLA_RT_cmac_b2cacc	3.8821	0	0
NV_NVDLA_CACC_regfile	3.1731	0	0
NV_NVDLA_CACC_assembly_ctrl	1.4798	0	0
NV_NVDLA_CACC_assembly_buffer	1.7694	0	0
NV_NVDLA_CACC_delivery_ctrl	2.7071	0	0
NV_NVDLA_CACC_delivery_buffer	1.3371	0	0
NV_NVDLA_CACC_calculator	0.1175	0	0
NV_NVDLA_cacc	-0.4724	-0.4724	-1097.3450
NV_NVDLA_partition_a	-0.2347	-0.2347	-118.1004

Tabella 4.6: Analisi post-sintesi di netlist generate da Yosys/ABC nell'ambiente Synopsys ad una frequenza operativa di $200MHz$

In questo caso, per quel che riguarda le strutture più complesse, si ottengono valori di slack negativi ad indicazione del fatto che la frequenza scelta risulta essere troppo elevata. Sebbene, alla luce di questi risultati, derivare un'indicazione generale sul confronto delle prestazioni dei due tool di sintesi sia impossibile, possono essere tratte alcune considerazioni:

- Per basse frequenze di clock, le netlist generate di Yosys/ABC risultano più efficienti in termini temporali di quelle ottenute da Synopsys Design Compiler, come i valori di CPS delle tabelle 5.1 e 5.2 indicano;

- Per frequenze di clock elevate l'andamento temporale risulta simile al caso con bassa frequenza ma esclusivamente per le strutture più semplici. Crescendo in complessità la situazione si capovolge, come si può vedere confrontando le ultime 3 righe delle tabelle 5.4 e 5.5.

Il secondo punto rivela uno dei limiti del FOSS di sintesi Yosys/ABC, ovvero l'incapacità di effettuare scelte di sintesi adeguate per strutture con un numero di gate elevato.
L'impossibilità di dedurre regole assolute sulle prestazioni dei due software si applica anche al caso del confronto dell'analisi temporale ottenuta nei due ambienti per le netlist generate da Yosys/ABC. In questo contesto, osservando le tabelle 5.1-5.3 e 5.4-5.6, i valori di CPS, WNS, TNS non coincidono esattamente, come ci si aspetterebbe. Per le entità semplici le differenze sono minime ed i due tool considerano gli stessi percorsi critici, come si può vedere dai seguenti due estratti di report relativi all'elemento NV_NVDLA_reset, il primo di Synopsys ed il secondo di OpenROAD.

Report 1 Analisi temporale su Synopsys del percorso critico per NV_NVDLA_reset a $2MHz$

```
Startpoint: _10_ (rising edge-triggered flip-flop clocked by nvdla_clk)
Endpoint: _11_ (rising edge-triggered flip-flop clocked by nvdla_clk)
Path Group: nvdla_clk
Path Type: max

Point                                          Fanout      Incr       Path
-------------------------------------------------------------------------
clock nvdla_clk (rise edge)                                0.0000     0.0000
clock network delay (ideal)                                0.0000     0.0000
_10_/CLK (sky130_fd_sc_hd__dfrtp_1)                        0.0000     0.0000
_10_/Q (sky130_fd_sc_hd__dfrtp_1)                          0.3478     0.3478
sync_reset_synced_rstn.NV_GENERIC_CELL.d0 (net)     1      0.0000     0.3478
_11_/D (sky130_fd_sc_hd__dfrtp_1)                          0.0000     0.3478
data arrival time                                                     0.3478

clock nvdla_clk (rise edge)                              500.0000   500.0000
clock network delay (ideal)                                0.0000   500.0000
_11_/CLK (sky130_fd_sc_hd__dfrtp_1)                        0.0000   500.0000
library setup time                                        -0.1093   499.8907
data required time                                                  499.8907
-------------------------------------------------------------------------
data required time                                                  499.8907
data arrival time                                                    -0.3478
-------------------------------------------------------------------------
slack (MET)                                                         499.5429
```

Report 2 Analisi temporale su OpenROAD del percorso critico per NV_NVDLA_reset a $2MHz$

```
Startpoint: _11_ (rising edge-triggered flip-flop clocked by nvdla_clk)
Endpoint: _10_ (rising edge-triggered flip-flop clocked by nvdla_clk)
Path Group: nvdla_clk
Path Type: max

Fanout     Cap    Slew   Delay    Time   Description
---------------------------------------------------------------------------
                  0.05    0.00    0.00   clock nvdla_clk (rise edge)
                          0.00    0.00   clock network delay (ideal)
                  0.05    0.00    0.00 ^ _11_/CLK (sky130_fd_sc_hd__dfrtp_1)
                  0.04    0.35    0.35 v _11_/Q (sky130_fd_sc_hd__dfrtp_1)
     1    0.00                           sync_reset_synced_rstn.NV_GENERIC_CELL.d0 (net)
                  0.04    0.00    0.35 v _10_/D (sky130_fd_sc_hd__dfrtp_1)
                                  0.35   data arrival time

                  0.05  500.00  500.00   clock nvdla_clk (rise edge)
                          0.00  500.00   clock network delay (ideal)
                          0.00  500.00   clock reconvergence pessimism
                                500.00 ^ _10_/CLK (sky130_fd_sc_hd__dfrtp_1)
                         -0.11  499.89   library setup time
                                499.89   data required time
---------------------------------------------------------------------------
                                499.89   data required time
                                 -0.35   data arrival time
---------------------------------------------------------------------------
                                499.54   slack (MET)
```

Per quel che riguarda invece le strutture più complesse, sebbene i valori di CPS non differiscano molto, i percorsi critici analizzati risultano diversi, come i valori di TNS nel caso con frequenza operativa di $200MHz$ mostrano. In Appendice C sono riportati i due estratti di report relativi alla struttura NV_NVDLA_cacc.
In conclusione, dall'analisi dei report temporali si è in grado di dire che la sintesi effettuata dal FOSS facente parte del flusso OpenROAD è in grado di fornire una netlist la cui struttura risulti comparabile in termini di timing con quella ottenuta utilizzando un software commerciale come Synopsys Design Compiler.

4.2 Analisi di potenza

La stima della potenza dissipata da un IC ottenuta dopo la fase di sintesi, sebbene dia un valore approssimativo dell'effettiva richiesta energetica del progetto, risulta necessaria al fine di poter effettuare potenziali ottimizzazioni pre-placement capaci di ridurre il consumo del progetto.
Come noto, la potenza che un circuito dissipa può essere raggruppata in due macro aree: la potenza statica e quella dinamica. Provando a dare una breve descrizione delle tipologie citate:

- Potenza statica: è la potenza dissipata dalla porta logica quando essa non è attiva, ovvero quando non vi è una variazione di valore in ingresso al gate. É quindi quel valore di potenza legato alle perdite (leakage) della porta (perdite source-drain sottosoglia, perdite tra lo strato di diffusione e il sub-strato). Tendenzialmente presenta valore estremamente piccoli se confrontati con la potenza dissipata di natura dinamica;

- Potenza dinamica: è la potenza dissipata dalla porta logica quando vi è una variazione del livello delle tensioni in ingresso ad essa, quindi durante lo stato di attività del gate. Dal momento che non è detto che una variazione di ingressi generi una variazione in uscita, la potenza dinamica viene a sua volta suddivisa in potenza interna e potenza di commutazione (switching power):

  - Potenza interna: è la dissipazione in condizione di non variazione dell'uscita della porta, quindi dovuta alla carica e scarica di tutte le capacità interne alla cella. In realtà il concetto di potenza interna include anche quella dissipata in condizione di cortocircuito tra il transistor P e quello N in fase di commutazione;

  - Potenza di commutazione: è la potenza dissipata dalla carica e scarica della capacità di carico totale. Oltre che dal valore effettivo del carico, la potenza di commutazione dipende dal tasso di carica/scarica di tale capacità e quindi, in ultima analisi, dalla frequenza di lavoro della cella.

Nelle seguenti tabelle sono stati raggruppati i risultati di questa analisi di potenza nelle stesse condizioni di lavoro del paragrafo precedente. Le tabelle 5.7-5.8-5.9 sono relative ad una frequenza operativa di $2MHz$ mentre le 5.10-5.11-5.12 ad una di $200MHz$.
In tali tabelle sono riportati i valori di potenza dinamica totale, quindi della somma di quella interna e quella di commutazione, e di potenza statica totale, definita "Leakage Power" poichè si riferisce esclusivamente a fenomeni di perdità.

	Total Dynamic Power[mW]	Leakage Power[nW]
NV_NVDLA_reset	0.9329e-04	3.34e-02
NV_NVDLA_sync3d	1.3541e-04	2.53e-02
NV_NVDLA_sync3d_s	1.4493e-04	2.37e-02
NV_NVDLA_RT_cmac_b2cacc	0.2529	67.2
NV_NVDLA_CACC_regfile	2.615e-02	9.09
NV_NVDLA_CACC_assembly_ctrl	2.392e-02	9.87
NV_NVDLA_CACC_assembly_buffer	0.8824	245
NV_NVDLA_CACC_delivery_ctrl	0.773	150
NV_NVDLA_CACC_delivery_buffer	0.3175	87.3
NV_NVDLA_CACC_calculator	3.63	1.41e03
NV_NVDLA_cacc	5.866	1.83e03
NV_NVDLA_partition_a	5.896	1.88e03

Tabella 4.7: Analisi post-sintesi di netlist generate da Yosys/ABC nell'ambiente OpenROAD ad una frequenza operativa di $2MHz$

	Total Dynamic Power[mW]	Leakage Power[nW]
NV_NVDLA_reset	1.8939e-04	5.7318e-02
NV_NVDLA_sync3d	2.5592e-04	5.4689e-02
NV_NVDLA_sync3d_s	2.6318e-04	5.7726e-02
NV_NVDLA_RT_cmac_b2cacc	0.3912	48.7610
NV_NVDLA_CACC_regfile	3.9779e-02	9.2664
NV_NVDLA_CACC_assembly_ctrl	0.1518	37.9497
NV_NVDLA_CACC_assembly_buffer	1.5766	380.9676
NV_NVDLA_CACC_delivery_ctrl	0.5465	106.5922
NV_NVDLA_CACC_delivery_buffer	0.5607	237.1902
	DW_lsd | NV_DW_lsd	
NV_NVDLA_CACC_calculator	2.8357 | 2.8368	1.7984e03 | 1.8022e03
NV_NVDLA_cacc	7.7191 | 7.7179	2.5607e03 | 2.2980e03
NV_NVDLA_partition_a	8.7299 | 8.7286	2.6094e03 | 2.3470e03

Tabella 4.8: Analisi post-sintesi di netlist generate da Synopsys Design Compiler nell'ambiente Synopsys ad una frequenza operativa di $2MHz$

	Total Dynamic Power[mW]	Leakage Power[mW]
NV_NVDLA_reset	1.7879e-04	4.5251e-02
NV_NVDLA_sync3d	2.5222e-04	2.5318e-02
NV_NVDLA_sync3d_s	2.6593e-04	3.1498e-02
NV_NVDLA_RT_cmac_b2cacc	0.3709	77.2135
NV_NVDLA_CACC_regfile	3.8649e-02	10.8902
NV_NVDLA_CACC_assembly_ctrl	1.6327e-02	11.1519
NV_NVDLA_CACC_assembly_buffer	1.3731	282.8176
NV_NVDLA_CACC_delivery_ctrl	0.5636	156.4829
NV_NVDLA_CACC_delivery_buffer	0.4773	96.9796
NV_NVDLA_CACC_calculator	2.6006	1.3810e03
NV_NVDLA_cacc	6.7571	1.6784e03
NV_NVDLA_partition_a	7.6581	1.6937e03

Tabella 4.9: Analisi post-sintesi di netlist generate da Yosys/ABC nell'ambiente Synopsys ad una frequenza operativa di $2MHz$

Come nella trattazione relativa all'analisi temporale, anche in questo caso non può essere fatta una stima generale su quale dei due sintetizzatori fornisca prestazioni migliori. Ciò che si può con certezza affermare è la conformità dei risultati.
Dato sicuramente interessante riguarda la potenza statica che le netlist dissipano. Come il confronto tra la tabella 5.7 e 5.8 può far vedere, tendenzialmente le netlist generate da Yosys/ABC tendono a aver un minor consumo statico rispetto a quelle ottenute su Synopsys Design Compiler. La ragione di tale risultato, al di là di algoritmi di stima di potenza differenti, si può ricercare nelle differenti aree che i vari elementi sintetizzati nei due ambienti occupano. Come verrà evidenziato nel successivo paragrafo, confrontando le dimensioni delle varie netlist generate dai due tool, l'ambiente open-source è in grado di fornire strutture a minor area, da cui ne segue un minor numero di celle utilizzate ed infine una minor potenza statica dissipata.
Ai fini del confronto dei risultati di sintesi, risulta interessante osservare le tabelle 5.7 e 5.9 relative alle analisi post sinstesi effettuate nei due ambiente della stessa netlist ottenuta da Yosys/ABC. Anche in questo caso, in analogia con l'analisi temporale, i due risultati non sono identici. Risulta quindi evidente come, nonostante i due tool lavorino con la stessa struttura sintetizzata, gli algoritmi di stima di potenza ed i contributi considerati per ottenere tale stima siano differenti.
Le successive tre tabelle mostrano i risultati di potenza per una frequenza operativa maggiore, $200MHz$. Come noto, l'unica tipologia di potenza che ha una dipendenza diretta con la frequenza di lavoro è la potenza dinamica poichè, come detto in precedenza, essa dipende dal tasso di carica/scarica delle varie capacità presenti, interne o esterne.
La riduzione del periodo di clock genera, quindi, un sostanziale aumento di tale tasso e, in conclusione, della potenza dinamica totale dissipata.
La potenza statica varia esclusivamente nel caso delle netlist generate da Synopsys Design Compiler ed indica la capacità del sintetizzatore di variare la quantità ed il tipo di

celle utilizzate al fine di ottenere il miglior risultato di sintesi in funzione della frequenza operativa. Tale modifica riguarda esclusivamente le strutture più complesse.
Di contro Yosys/ABC non tiene conto delle variazioni del periodo di clock producendo la stessa netlist, come risulta evidente dai valori di potenza statica costanti per le due frequenze. Questo è chiaramente un limite del tool open-source che rende il flusso OpenROAD poco versatile.

	Total Dynamic Power[mW]	Leakage Power[mW]
NV_NVDLA_reset	0.9329e-02	3.34e-02
NV_NVDLA_sync3d	1.3541e-04	2.53e-02
NV_NVDLA_sync3d_s	1.4493e-02	2.37e-02
NV_NVDLA_RT_cmac_b2cacc	25.29	67.2
NV_NVDLA_CACC_regfile	2.615	9.09
NV_NVDLA_CACC_assembly_ctrl	2.392	9.87
NV_NVDLA_CACC_assembly_buffer	88.24	245
NV_NVDLA_CACC_delivery_ctrl	77.3	150
NV_NVDLA_CACC_delivery_buffer	31.75	87.3
NV_NVDLA_CACC_calculator	363	1.41e03
NV_NVDLA_cacc	586.6	1.83e03
NV_NVDLA_partition_a	589.6	1.88e03

Tabella 4.10: Analisi post-sintesi di netlist generate da Yosys/ABC nell'ambiente OpenROAD ad una frequenza operativa di $200MHz$

	Total Dynamic Power[mW]	Leakage Power[mW]
NV_NVDLA_reset	1.8939e-02	5.7318e-02
NV_NVDLA_sync3d	2.5592e-02	5.4689e-02
NV_NVDLA_sync3d_s	2.6318e-02	5.7726e-02
NV_NVDLA_RT_cmac_b2cacc	39.1258	48.7610
NV_NVDLA_CACC_regfile	3.9779	9.2664
NV_NVDLA_CACC_assembly_ctrl	15.1823	37.9682
NV_NVDLA_CACC_assembly_buffer	157.6457	381.1136
NV_NVDLA_CACC_delivery_ctrl	54.6619	106.8049
NV_NVDLA_CACC_delivery_buffer	56.1769	237.1138
	DW_lsd | NV_DW_lsd	
NV_NVDLA_CACC_calculator	283.4487 | 280.2684	1.7992e03 | 1.5854e03
NV_NVDLA_cacc	771.7706 | 771.7484	2.6464e03 | 2.2759e03
NV_NVDLA_partition_a	876.5197 | 876.4471	2.6521e03 | 2.3268e03

Tabella 4.11: Analisi post-sintesi di netlist generate da Synopsys Design Compiler nell'ambiente Synopsys ad una frequenza operativa di $200MHz$

	Total Dynamic Power[mW]	Leakage Power[mW]
NV_NVDLA_reset	1.7879e-02	4.5251e-02
NV_NVDLA_sync3d	2.5222e-02	2.5318e-02
NV_NVDLA_sync3d_s	2.6593e-02	3.1498e-02
NV_NVDLA_RT_cmac_b2cacc	37.0906	77.2135
NV_NVDLA_CACC_regfile	3.8649	10.8902
NV_NVDLA_CACC_assembly_ctrl	1.6327	11.1519
NV_NVDLA_CACC_assembly_buffer	137.5425	282.8342
NV_NVDLA_CACC_delivery_ctrl	56.3614	156.4829
NV_NVDLA_CACC_delivery_buffer	47.8245	97.0198
NV_NVDLA_CACC_calculator	260.060	1.3810e03
NV_NVDLA_cacc	675.7112	1.6784e03
NV_NVDLA_partition_a	765.8081	1.6937e03

Tabella 4.12: Analisi post-sintesi di netlist generate da Yosys/ABC nell'ambiente Synopsys ad una frequenza operativa di $200MHz$

I risultati elencati nelle tabelle 5.10-5.11-5.12 confermano quanto enunciato precedentemente.
Le considerazioni relative ai differenti valori dissipazione per le netlist generate dai due tool ed analizzate nei due ambienti sono identiche a quelle fatte nel caso di un periodo di clock di $500ns$.

4.3 Analisi dell'area

La stima post-sintesi dell'area occupata da un IC, seppur semplicemente indicativa, è fondamentale durante la fase di progettazione poichè da un'idea di quali possano essere le dimensioni fisiche di cui la parte "logica" del design necessita. Da questa stima ne derivano le dimensioni effettive del chip che il progettista deve stabilire affinchè, durante la fase di P&R, non vi siano DRV e problemi di congestione.
Nelle tabelle di seguito riportate sono mostrate le aree espresse in μm^2 delle netlist generate da Yoysy/ABC e da Synopsys Design Compiler. Ricordando i tre amibienti di lavoro:

- Analisi post-sintesi di netlist generate da Yosys/ABC nell'ambiente OpenROAD;
- Analisi post-sintesi di netlist generate da Synopsys Design Compiler nell'ambiente Synopsys;
- Analisi post-sintesi di netlist generate da Yosys/ABC nell'ambiente Synopsys;

la prima colonna fa riferimento al primo scenario e di seguito le altre. Si può vedere come, in genere, le netlist generate dal FOSS occupino una minor area. Come accennato in precedenza questa condizione si ripercuote in una minore dissipazione di potenza statica.

	Area dei design[μm^2]		
NV_NVDLA_reset	73	$2MHz$: 96.3424 $200MHz$: 96.3424	72.5696
NV_NVDLA_sync3d	60	$2MHz$: 91.3376 $200MHz$: 91.3376	60.0576
NV_NVDLA_sync3d_s	79	$2MHz$: 110.1056 $200MHz$: 110.1056	78.8256
NV_NVDLA_RT_cmac_b2cacc	147544	$2MHz$: 133942.2103 $200MHz$: 133942.2103	147544.0019
NV_NVDLA_CACC_regfile	22154	$2MHz$: 20184.3584 $200MHz$: 20184.3584	22153.7470
NV_NVDLA_CACC_assembly_ctrl	17268	$2MHz$: 56671.8527 $200MHz$: 56728.1567	17267.8108
NV_NVDLA_CACC_assembly_buffer	510600	$2MHz$: 851322.7147 $200MHz$: 851529.1626	510599.6896
NV_NVDLA_CACC_delivery_ctrl	393385	$2MHz$: 286731.2432 $200MHz$: 287311.8000	393384.7761
NV_NVDLA_CACC_delivery_buffer	201526	$2MHz$: 468384.2078 $200MHz$: 467956.2974	201525.7764

Tabella 4.13: Analisi post-sintesi delle aree degli elementi facenti parte della gerarchia di NV_NVDLA_partition_a

	Area dei design[μm^2]		
NV_NVDLA_CACC_calculator	3087545	$2MHz$: 4102278.0835 $200MHz$: 4104530.2436	3087544.8631
NV_NVDLA_cacc	4156586	$2MHz$: 5794196.9817 $200MHz$: 5828572.4365	4156586.3869
NV_NVDLA_partition_a	4272348	$2MHz$: 5926268.6481 $200MHz$: 5888537.4477	4272347.4077

Tabella 4.14: Analisi post-sintesi delle aree degli elementi facenti parte della gerarchia di NV_NVDLA_partition_a che utilizzano il blocco DW_lsd

	Area dei design[μm^2]		
NV_NVDLA_CACC_calculator	3087545	$2MHz$: 4143546.4124 $200MHz$: 3585466.1762	3087544.8631
NV_NVDLA_cacc	4156586	$2MHz$: 5276012.5079 $200MHz$: 5114129.7407	4156586.3869
NV_NVDLA_partition_a	4272348	$2MHz$: 5409101.3998 $200MHz$: 5247326.2358	4272347.4077

Tabella 4.15: Analisi post-sintesi delle aree degli elementi facenti parte della gerarchia di NV_NVDLA_partition_a che utilizzano il blocco NV_DW_lsd

In conclusione di questo paragrafo può essere rilevante analizzare come i due tool riportino le informazioni relative all'area, ovvero quali informazioni diano su come essa sia ripartita tra i vari elementi interni ai blocchi.
Mentre il tool open-source riporta esclusivamente il valore netto dell'area occupata, senza alcun particolare, i report di Synopsys Design Compiler, al di là del totale, contengono maggiori informazioni, come il seguente estratto mostra:

Report 3 Estratto relativo all'area del report ottenuto su Synopsys Design Compiler del blocco NV_NVDLA_parition_a a $200MHz$

```
Area
-----------------------------------
Combinational Area:    2431303.0238
Noncombinational Area: 2816023.2120
Buf/Inv Area:           330717.1738
Total Buffer Area:       36337.3493
Total Inverter Area:    294379.8245
Macro/Black Box Area:        0.0000
Net Area:                    0.0000
-----------------------------------
Cell Area:             5247326.2358
Design Area:           5247326.2358
```

Risulta evidente come una maggiore chiarezza sulla ripartizione dell'area possa aiutare il progettista nel decidere eventuali variazioni da attuare sul design al fine di ottenere un risultato più efficiente.

Finita l'analisi dei risultati di sintesi si passerà, nel prossimo capitolo, a quella relativa al P&R. In un capitolo successivo, invece, verranno presentate le potenzialità e le criticità riscontrate durante il lavoro di sintesi utilizzando gli strumenti del flusso OpenROAD.

Capitolo 5

P&R

In questo capitolo verranno mostrati e commentati i risultati di P&R di diverse sotto-strutture interne alla NV_NVDLA_partiton_a ottenuti sia in ambiente OpenROAD che utilizzando il software proprietario Cadence Innovus Implementation System.
Le metriche di confronto sono le stesse utilizzate nel capitolo precedente, ovvero timing, potenza ed area occupata dal design. In questo caso, al fine di ottenere un confronto il più esaustivo possibile, tali dati verrano analizzati in tre momenti differenti del flusso di P&R: post-placement, post-CTS e post-routing. Questo darà la possibilità di valutare l'efficacia dei due ambienti di lavoro.
A differenza dell'analisi realizzata durante il passo di sintesi, verranno comparate, per ogni elemento, i risultati di P&R esclusivamente delle netlist generate dal FOSS Yosys/ABC. Dal momento che questo lavoro di tesi si fonda sul confronto dei risultati dei due ambienti di lavoro, open-source e con licenza, effettuare il P&R delle netlist ottenute tramite Synopsys Design Compiler risulta essere superflua.
Si annota, in questa premessa, che vi è stata la necessità di ridurre maggiormente, anche rispetto al capitolo inerente la sintesi, il numero di elementi facenti parte della partizione dell'accelleratore sotto analisi. Nel dettaglio sono stati effettuati i P&R dei seguenti componenti:

- NV_NVDLA_reset;
- NV_NVDLA_sync;
- NV_NVDLA_sync_3d;
- NV_NVDLA_CACC_regfile.

Come verrà analizzato in maggior dettaglio nel successivo capitolo relativo alle potenzialità e alle criticità del flusso OPENRoad, la complessità degli altri blocchi facente parte della NV_NVDLA_partiton_a, mostrati in figura 5.1, ed il conseguente carico computazionale necessario per completare la generazione del file GDSII sono risultati eccessivi per l'ambiente open-source.
Anche il P&R è stato effettuato sotto due regimi di clock differenti, $2MHz$ e $200MHz$, generando delle differenze a livello di CTS, come verrà analizzato nel paragrafo associato.

Altro accenno ai limiti del flusso open-source, che verrà investigato maggiormente nel successivo capitolo, riguarda l'allocazione, durante la fase di floorplan, dell'area utilizzabile dall'elemento sotto analisi. Di fatto il P&R effettuato in ambiente open-source inizializza ed utilizza un'area nettamente maggiore rispetto ai risultati ottenibili su Cadence Innovus Implementation. La ragione di tale discrepanza è stata ritrovata nella difficoltà da parte dei FOSS di inizializzare un'adeguata area di placement sulla base dell'effettiva area necessaria alla netlist avendo definito durante il floorplan i parametri di CORE_UTILIZZATION e PLACE_DENSITY, ovvero la percentuale di area del die destinata alla netlist e quanto ,su questa percentuale, vengano sparse le celle. Si rimanda al capitolo relativo alle difficoltà di uso del flusso OpenROAD per un'analisi più approfondita.

Seguono, in questo capitolo, i risultati di P&R nei tre passi cardine.

5.1 Analisi temporale

Le tabelle 6.1 e 6.2 riportano i risultati di analisi temporale, in termini di *Setup time* e *Hold time*, dei due ambienti di lavoro in rifermento a tre diversi step del processo di P&R: post-placement, post-CTS e post-ROUTE. Le frequenze operative scelte sono le medesime usate durante il passo di sintesi, ovvero 2MHz e 200MHz.
La terza colonna presente nelle due tabelle riporta i risultati in una condizione particolare, ovvero con un vincolo, su Cadence Innovus Implementation System, sulla scelta delle celle da utilizzare nella realizzazione dell'albero di clock. La necessità di questa limitazione deriva dal fatto che il tool presente nel flusso OpenROAD per la sintesi dell'albero di clock, TritonCTS, non è in grado di effettuare una scelta ottimale sulla base di un insieme di celle ultilizzabili, ma si limita ad usare un unico elemento. In questo caso la cella di buffer utilizzata è la $sky130\_fd\_sc\_hd\_\_buf\_1$. Nella seconda colonna tale costrizione non è stata impostata ed il software di Cadence ha potuto effettuare le scelte migliori in tal senso.
Per concludere il discorso sui vincoli relativi alla CTS, nel caso di strutture più semplici, ovvero le prime tre, Innovus genera una rete di clock anche in assenza di buffer appositi. Discorso analogo non può invece essere fatto per TritonCTS, il quale genera una rete dell'albero di clock sempre con la presenza di buffer. Le figure 6.1 e 6.2 mostrano le celle piazzate post CTS dai due tool, Innovus e TritonCTS rispettivamente, per il blocco NV_NVDLA_reset.

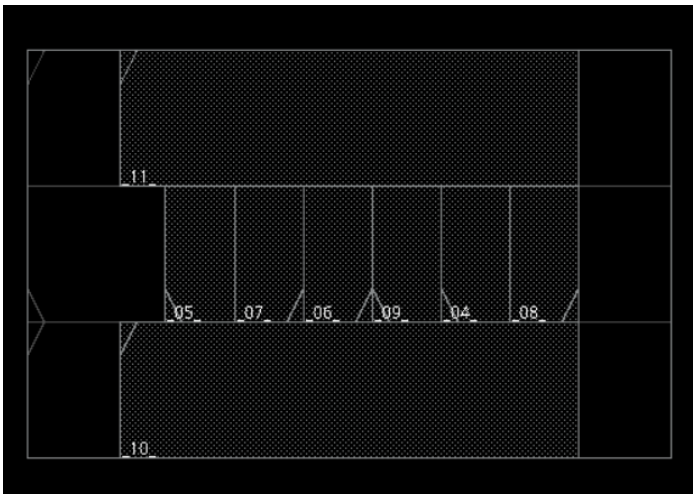


Figura 5.1: Celle utilizzate da Cadence Innovus Implementation System post-CTS di NV_NVDLA_reset

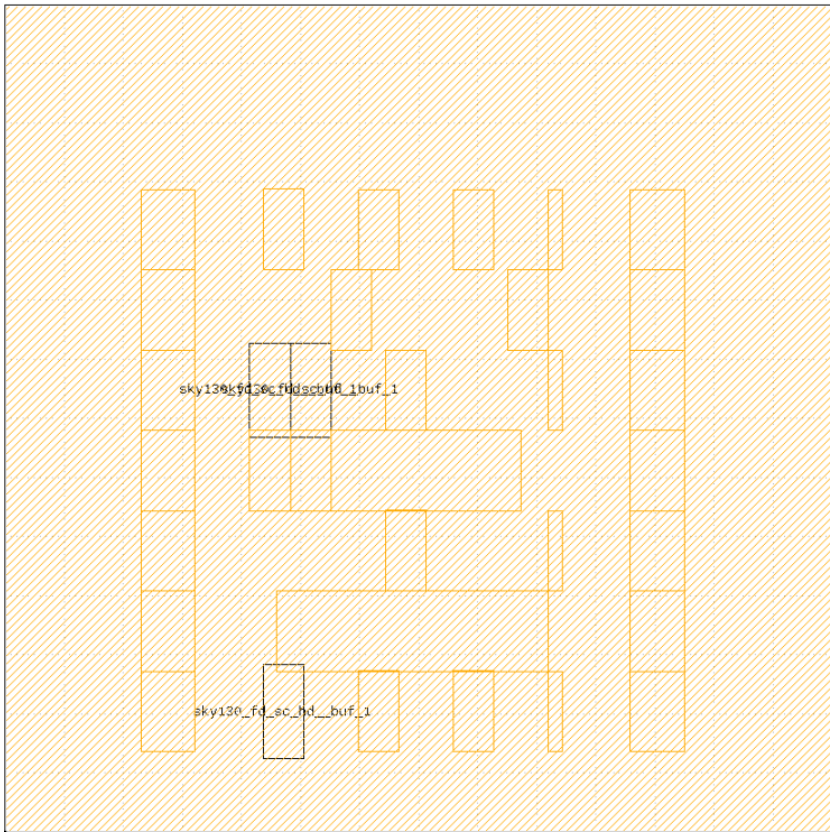


Figura 5.2: Celle utilizzate da TritonCTS post-CTS di NV_NVDLA_reset

In figura 5.2 sono state evidenziate le celle utilizzate per sintesi dell'albero di clock.

Questo "limite" ha sia effetti positivi, cioè un aumento dei Setup e Hold Time rispetto ad Innovus (vedere le righe corrispettive dei primi tre elementi), ma, come si vedrà nel capitolo relativo alle stime di potenza, anche degli effetti negativi. Risulta chiaro come l'inserimento di tali celle nel flusso open-source generi un consumo maggiore.
L'analisi dei risultati in termini temporali post-ROUTE presenta un deterioramento dei valori dato dall'aumento del carico capacitivo presente sui percorsi critici, come è ovvio che sia.

		Slack(*ns*)					
		OpenROAD		*Innovus*		*Innovus*(*buff cons*)	
		Setup	Hold	Setup	Hold	Setup	Hold
post-PLACE	NV_NVDLA_reset	499,54	0,35	499,527	0,358	–	–
	NV_NVDLA_sync3d	499,61	0,33	499,605	0,326	–	–
	NV_NVDLA_sync3d_s	499,44	0,34	499,433	0,354	–	–
	NV_NVDLA_CACC_regfile	497,93	0,35	487,794	0,032	487,794	0,032
post-CTS	NV_NVDLA_reset	499,54	0,35	499,531	0,349	–	–
	NV_NVDLA_sync3d	499,59	0,31	499,609	0,319	–	–
	NV_NVDLA_sync3d$_s$	499,44	0,34	499,442	0,349	–	–
	NV_NVDLA_CACC_regfile	497,23	0,35	495,025	-0,082	495,095	0,001
post-ROUTE	NV_NVDLA_reset	499,53	0,36	499,533	0,348	–	–
	NV_NVDLA_sync3d	499,57	0,33	499,611	0,315	–	–
	NV_NVDLA_sync3d$_s$	499,42	0,35	499,443	0,35	–	–
	NV_NVDLA_CACC_regfile	493,32	0,23	494,897	-0,088	494,932	-0,009

Tabella 5.1: Risultati di timing, in termini di setup e hold time, durante il P&R ad una frequenza operativa di 2MHz

Analisi analoga a quella effettuata nel caso di frequenza operativa di 2MHz si può applicare anche al caso di una frequenza di 200MHz.

		Slack(*ns*)					
		OpenROAD		*Innovus*		*Innovus*(*buff cons*)	
		Setup	Hold	Setup	Hold	Setup	Hold
post-PLACE	NV_NVDLA_reset	4,54	0,35	4,527	0,358	–	–
	NV_NVDLA_sync3d	4,61	0,33	4,605	0,326	–	–
	NV_NVDLA_sync3d_s	4,45	0,36	4,433	0,354	–	–
	NV_NVDLA_CACC_regfile	2,93	0,35	-7,206	0,032	-7,26	0,032
post-CTS	NV_NVDLA_reset	4,54	0,35	4,531	0,349	–	–
	NV_NVDLA_sync3d	4,59	0,31	4,609	0,319	–	–
	NV_NVDLA_sync3d$_s$	4,44	0,34	4,441	0,0349	–	–
	NV_NVDLA_CACC_regfile	2,23	0,35	0,096	-0,082	0,0186	0,001
post-ROUTE	NV_NVDLA_reset	4,52	0,36	4,529	0,352	–	–
	NV_NVDLA_sync3d	4,57	0,33	4,609	0,318	–	–
	NV_NVDLA_sync3d$_s$	4,42	0,34	4,441	0,351	–	–
	NV_NVDLA_CACC_regfile	-1,68	0,23	-0,041	-0,094	0,06	-0,006

Tabella 5.2: Risultati di timing, in termini di setup e hold time, durante il P&R ad una frequenza operativa di 200MHz

Come nel capitolo relativo alla sintesi, anche per quel che riguarda il P&R non è possibile dedurre un andamento assoluto sulle prestazioni dei due flussi.
I due tool considerano percorsi critici differenti, a controprova del fatto che effettuino delle stime basandosi su aolgoritmi ed elementi diversi diversi.
Ciononostante, i risultati relativi all'analisi temporate post-ROUTE sembrano suggerire come le scelte effettuate dal software proprietario portino ad un risultato migliore.

5.2 Analisi di area

In tabella 6.3 sono mostrati le aree occupate, in termini di μm^2, dagli elementi analizzati nei tre step di P&R.
Come anticipato nella prima parte di questo capitolo, il flusso OpenROAD presenta dei valori maggiori rispetto a quelli ottenuti utilizzando Innovus, differenza che si acuisce nello step post-ROUTE. Tale incremento sostanziale, anche confrontando i risultati post-CTS, viene giustificato dal fatto che i tool open-source tengono conto anche dell'area occupata da elementi di riempimento, quali ad esempio sky130_fd_sc_hd___fill, come la figura 5.3 mostra, i quali non hanno nessuna funzione logica ma sono utilizzate ed inserite esclusivamente per garantire la continuità della NWELL, ed evitare, durante la fase di DRC (Design Rule Check) errori della forma "NWell minimum spacing not met".

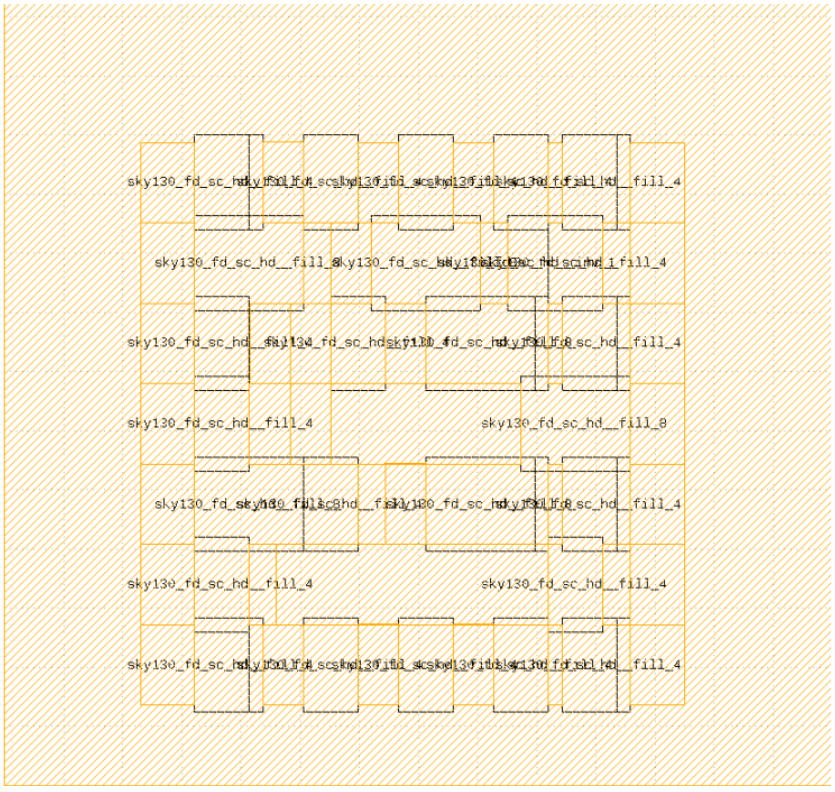


Figura 5.3: Dettaglio delle celle filler inserite nel flusso OPERNRoad di NV_NVDLA_reset

In figura 5.4 si può, invece, notare i filler inseriti in l'ambiente proprietario

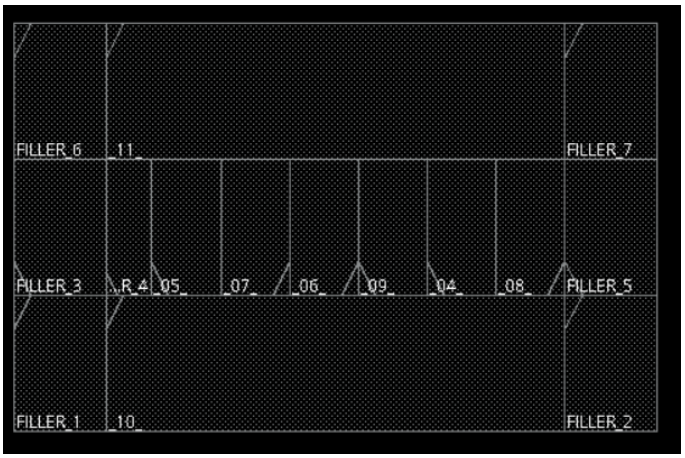


Figura 5.4: Dettaglio delle celle filler inserite nel flusso Cadence Innovus Implementation System NV_NVDLA_reset

Dal confronto delle due immagini si può notare come le scelte di placement effettuate da Innovus risultano evidentemente migliori.
Il problema relativo ai filler inseriti da OPENRoad si configura come un grosso limite dell'ambiente di lavoro open-source, poichè risulta evidente come si perda cognizione dell'effettivo utilizzo di area dell'elemento sotto analisi.
Un'altra considerazione rilevante si può ottenere confrontando i risultati di area post sintesi, in tabella 5.13, con quelli della tabella 6.3. Si nota, infatti, come le aree occupate dai design implementati su Cadence Innovus Implementation System siano identici a quelli generati da Synopsys Design Compiler. In tale senso il P&R del software proprietario è consistente. Tale considerazione non può invece essere applicata a flusso OpenROAD.
Come anticipato nella parte introduttiva di questo capitolo, i tool open-source, forniti i parametri di CORE_UTILIZZATION e PLACE_DENSITY, restituiscono dei valori di area estremamente elevati, inficiando totalmente il lavoro di comparazione. Nella definizione dell'area occupabile dal design, nel flusso OpenROAD, si è quindi ragionato in maniera diversa. Sono state definite le coordnate di CORE_AREA e DIE_AREA, espresse entrambe nella forma di coordinate cartesiane $\{dx; dy\}$.
Tale approccio, sebbene garantisca al progettista maggior controllo nella fase di definizione dell'area dell'IC, utilizzato nel flusso OpenROAD raggiunge un punto di minimo, abbastanza elevato, oltre il quale i tool non sono in grado di convergere, specialmente nella fase di routing, portando a dei risultati nettamente inferiori se confrontati con quelli ottenuti sul software proprietario. In aggiunta, l'approccio utilizzato per ottenere questo minimo è stato di tipo *trial and error*. Sono stati via via impostate coordinate sempre minori al fine di raggiungere il punto limite di non convergenza. Risulta ovvio come tale metodologia non sia ottimale in termini di tempi di progettazione. Bisogna ricordare come le strutture analizzate siano di una certa semplicità, ad indicazione del fatto che il flusso OpenROAD non sia ancora in grado di affrontare le sfide che gli attuali IC, i quali

presentano un tasso di complessità strutturale elevato, offrono.

		$Area(\mu m^2)$		
		OpenROAD	*Innovus*	*Innovus*(*buff cons*)
post-PLACE	NV_NVDLA_reset	163	72,5696	–
	NV_NVDLA_sync3d	128	60,0576	–
	NV_NVDLA_sync3d_s	165	78,8256	–
	NV_NVDLA_CACC_regfile	26577	21265,3952	21265,3952
post-CTS	NV_NVDLA_reset	174	72,5696	–
	NV_NVDLA_sync3d	139	60,0576	–
	NV_NVDLA_sync3d$_s$	176	78,8256	–
	NV_NVDLA_CACC_regfile	26663	21423,0464	21479,3504
post-ROUTE	NV_NVDLA_reset	354	72,5696	–
	NV_NVDLA_sync3d	248	60,0576	–
	NV_NVDLA_sync3d$_s$	324	78,8256	–
	NV_NVDLA_CACC_regfile	73445	21423,0464	21479,3504

Tabella 5.3: Report delle aree occupate dalle istanze analizzate durante il P&R

Come si vedrà nel successivo paragrafo, relativo all'analisi della potenza dissipata dalle varie istanze, l'elevato valore in termini di area del P&R effettuato con gli strumenti open-source non si traduce in un "esplosione" della dissipazione. Di fatto le celle con funzione logica utilizzate, le quali sono le sole a dare un contributo di potenza, sono identiche, a meno di quelle utilizzate come buffer dall'albero di clock. Questo si traduce in consumi comparabili, se non addirittura migliori nel caso open-source. Segue, quindi, tale analisi di potenza.

5.3 Analisi di potenza

In tabella 6.4 sono riportati i valori di potenza totale dissipata dagli elementi processati, ad una frequenza operativa di 2MHz, nelle tre condizioni operative:

- Flusso OPENRoad;
- Ambiente Cadence Innovus Implementation System con limite sui buffer utilizzabili per la CTS;
- Ambiente Cadence Innovus Implementation System in assenza di un limite sui buffer utilizzabili per la CTS.

Differentemente dal capitolo relativo alla sintesi, si è deciso di riportare esclusivamente le stime di potenza totale, la quale è composta da tre componenti:

- Potenza interna;
- Potenza di commutazione;
- Potenza di perdita.

		$Total\ Power(mW)$		
		$OpenROAD$	$Innovus$	$Innovus(buff\ cons)$
post-PLACE	NV_NVDLA_reset	0,000097	0,0002111	–
	NV_NVDLA_sync3d	0,000138	0,00026836	–
	NV_NVDLA_sync3d_s	0,000144	0,00029199	–
	NV_NVDLA_CACC_regfile	0,0275	0,05225158	0,05225158
post-CTS	NV_NVDLA_reset	0,000266	0,00021403	–
	NV_NVDLA_sync3d	0,00032	0,00027372	–
	NV_NVDLA_sync3d$_s$	0,000321	0,00029546	–
	NV_NVDLA_CACC_regfile	0,0389	0,06141357	0,06275007
post-ROUTE	NV_NVDLA_reset	0,000301	0,0002138	–
	NV_NVDLA_sync3d	0,000363	0,00027326	–
	NV_NVDLA_sync3d$_s$	0,000363	0,00029525	–
	NV_NVDLA_CACC_regfile	0,0491	0,06136598	0,06278711

Tabella 5.4: Risultati di potenza totale dissipata dai blocchi analizzati durante il P&R ad una frequenza operativa di 2MHz

Come anticipato nei precedenti paragrafi, le prime tre istanze, una volta effettuata la sintesi dell'albero di clock in ambiente OPENRoad, presentano consumi maggiori dal momento che per esse il software proprietario è in grado di realizzare un albero privo di buffer o inverter, celle che presentano, generalmente, consumi elevati data la loro funzione. Andando, invece, a confrontare i risultati post-CTS e post-Route del blocco NV_NVDLA_CACC_regfile si può vedere come l'analisi effettuata in ambiente open-source presenti valori minori. Tale dato trova spiegazione andando ad analizzare il numero di celle utiizzate per la CTS dai due software.

Report 4 Estratto relativo al numero di buffer utlizzati da TritonCTS per la realizzazione dell'albero di clock per l'elemento NV_NVDLA_CACC_regfile

```
[INFO CTS-0018]     Created 23 clock buffers.
```

Report 5 Estratto relativo al numero di buffer utlizzati da Cadence Innovus Implementation System per la realizzazione dell'albero di clock per l'elemento NV_NVDLA_CACC_regfile senza constraint sulla cella da utilizzare

```
 Clock DAG stats at end of CTS:
  ==============================

  ------------------------------------------------------------
  Cell type                     Count    Area       Capacitance
  ------------------------------------------------------------
  Buffers                         0        0.000       0.000
  Inverters                       6      180.173       0.242
  Integrated Clock Gates          0        0.000       0.000
  Non-Integrated Clock Gates      0        0.000       0.000
  Clock Logic                     0        0.000       0.000
  All                             6      180.173       0.242
  ------------------------------------------------------------
```

Report 6 Estratto relativo al numero di buffer utlizzati da Cadence Innovus Implementation System per la realizzazione dell'albero di clock per l'elemento NV_NVDLA_CACC_regfile con constraint sulla cella da utilizzare

```
Clock DAG stats at end of CTS:
  ==============================

  ------------------------------------------------------------
  Cell type                     Count    Area       Capacitance
  ------------------------------------------------------------
  Buffers                        33      123.869       0.072
  Inverters                       0        0.000       0.000
  Integrated Clock Gates          0        0.000       0.000
  Non-Integrated Clock Gates      0        0.000       0.000
  Clock Logic                     0        0.000       0.000
  All                            33      123.869       0.072
  ------------------------------------------------------------
```

Sebbene le informazione ottenibili in su OpenRoad siano minime, si puo vedere come, confrontando il Report 4 con il Report 5, il flusso open-source utilizzi un minor numero di celle. Ciò giustifica i ridotti valori di dissipazione.
Per ciò che riguarda la terza colonna, considerando che la cella utlizzata da Innovus per realizzare la CTS, senza limitazioni, è la sky130_fd_sc_hd___clkinv_16, inverter di dimensioni elevate se confrontata alla sky130_fd_sc_hd___buf_1, come le figure 5.6 e 5.7

mostrano, i consumi maggiori possono essere giustificati.

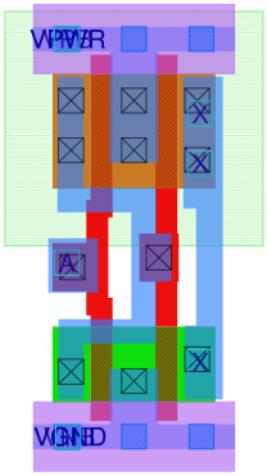


Figura 5.5: sky130_fd_sc_hd___buf_1

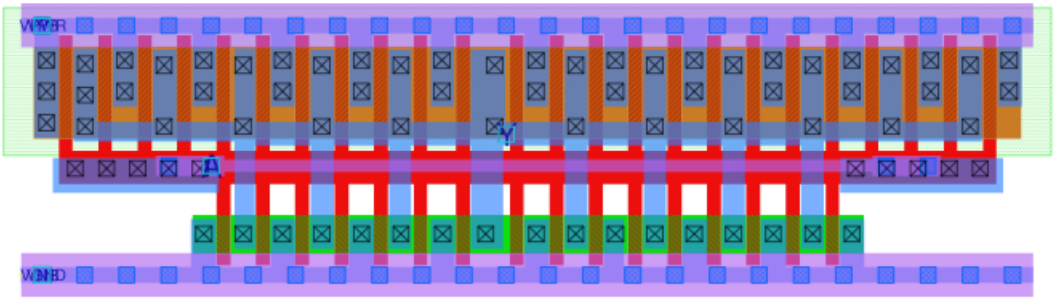


Figura 5.6: sky130_fd_sc_hd___clkinv_16

Nonostante ciò, risulta difficoltoso effettuare un'analisi completa dei flussi dal momento che i vari stimatori di potenza utilizzati sfruttano algoritmi differenti.
La tabella 6.5 mostra i risultati di dissipazione ad una frequenza maggiore rispetto che al caso precedente, di 200MHz.

		$Total\ Power(mW)$		
		$OpenROAD$	$Innovus$	$Innovus(buff\ cons)$
post-PLACE	NV_NVDLA_reset	0,00904	0,02110548	–
	NV_NVDLA_sync3d	0,0138	0,02683317	–
	NV_NVDLA_sync3d_s	0,0144	0,02919604	–
	NV_NVDLA_CACC_regfile	2,75	5,22402657	5,22402657
post-CTS	NV_NVDLA_reset	0,0266	0,02139817	–
	NV_NVDLA_sync3d	0,032	0,02736969	–
	NV_NVDLA_sync3d$_s$	0,0321	0,02954269	–
	NV_NVDLA_CACC_regfile	3,89	6,14356807	6,27755853
post-ROUTE	NV_NVDLA_reset	0,03	0,02140803	–
	NV_NVDLA_sync3d	0,0363	0,02736076	–
	NV_NVDLA_sync3d$_s$	0,0363	0,02957054	–
	NV_NVDLA_CACC_regfile	4,91	6,14028578	6,27558671

Tabella 5.5: Risultati di potenza totale dissipata dai blocchi analizzati durante il P&R ad una frequenza operativa di 200MHz

Come è stato analizzato nel capitolo precedente, relativo alla sintesi, una aumento della frequenza operativa comporta un incremento sostanziale della potenza dinamica. Si può vedere come tale andamento sia conprovato dai risultati mostrati in tabella.
L'analisi svolta in relazione al caso di periodo di clock più elevato, di 500ns, può essere ricondotta anche in questo caso con un periodo di 5ns.
Gli alberi di clock vengono sintetizzati in egual modo rispetto ai risultati in tabella 6.4, mostrando come l'unico effettivo cambiamento riguarda appunto la potenza dinamica.
Le ultime due immagini, 5.5 e 5.6, fanno vedere i risultati finali dei due flussi, open-source e proprietario rispettivamente:

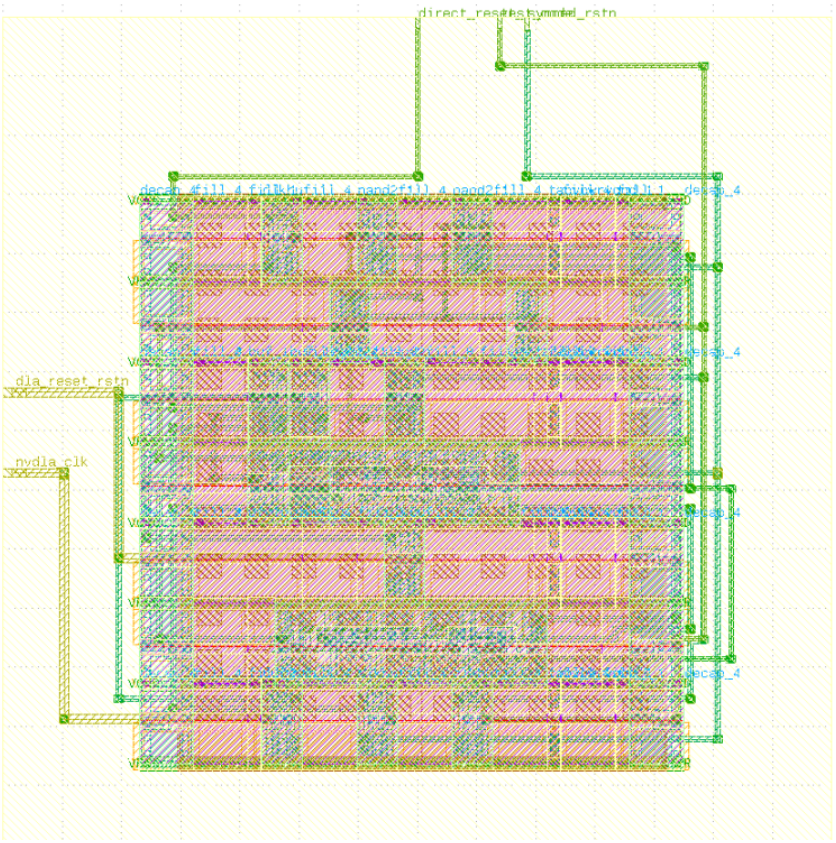


Figura 5.7: Risultato finale su OPENRoad di NV_NVDLA_reset

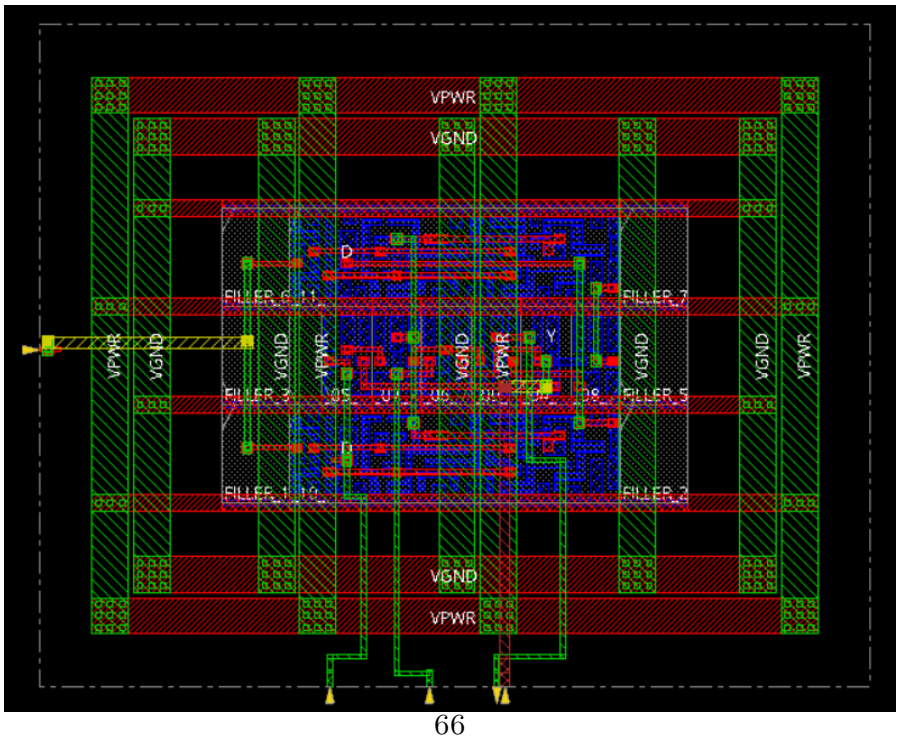


Figura 5.8: Risultato finale su Cadence Innovus Implementation System di NV_NVDLA_reset

Terminata questa parte di analisi e confronto di sintesi e P&R operati nei due ambienti di lavoro passeremo al capitolo finale in cui verranno, sinteticamente, riportati sia gli aspetti positivi del progetto OPENRoad sia quelli negativi. Alla luce di questo confronto si capirà se il flusso open-source è in grado di dimostrarsi uno strumento competitivo o meno.

Capitolo 6

Conclusione: pro e contro del flusso OPENRoad

In conclusione di questo lavoro di tesi, a seguito di tutto il lavoro di ricerca e di implementazione del flusso open-source, dopo avere analizzato e comparato i risultati di sintesi e P&R ottenuti nei due ambienti sotto studio siamo stati in grado di comprendere quali siano le potenzialità e le criticità del progetto OPENRoad.
Il capitolo che segue partità con il resoconto dei punti positivi che ci sentiamo di attribuire ai FOSS utilizzati e terminerà con quelle che sono le negatività ancora presenti, sempre in un lavoro di confronto con i vari tool proprietari utilizzati.

6.1 Potenzialità

La prima peculiarità positiva del progetto ad opera del DARPA è sicuramente la facilità d'uso del flusso da RTL a GDSII.
Una volta installati tutti i tool, viene data a disposizione dell'utente un Makefile il quale comprende e richiama tutti gli script utili all'implementazione dell'IC su cui si vuole lavorare. Di fatto il progettista dovrà esclusivamente settare alcuni parametri generali, quali ad esempio il codice sorgente da utilizzare, l'area del dispositivo, la tecnologia di riferimento ed il software OPENRoad provvederà alla realizzazione del design prefissato. Questa peculiarità sicuramente risponde positivamente ad uno dei principi cardine del progetto OPENRoad ovvero la "no human in the loop", consentendo quindi al progettista una certa "libertà lavorativa" maggiore, in quanto la sua presenza sarebbe strettamente necessaria solo in fase di inizializzazione del flusso.
Legato a questo aspetto vi è un'altra caratteristica positiva importante dell'ambiente open-source: la customizzazione del flusso. Il progettista, una volta individuati i punti cardine dei vari script, da quello di sintesi, a quello di placement, per finire con quello di routing, ha piena libertà di modifica senza precludere il corretto funzionamento dello stesso. Questa versatilità rende il progetto OPENRoad un potente strumento di implementazione di IC capace quindi di adattarsi a diversi scenari di progettazione.
Parlando invece dei risultati ottenuti nei precedenti capitoli si può vedere come, nel caso

di stutture di una discreta semplicità, quelli forniti dal flusso open-source siano confrontabili con quelli ottenuti dai due tool Cadence. Questo dato mostra quanto, in potenza, il progetto OPENRoad possa in futuro diventare una valida alternativa nella progettazione di IC.
In un'ottica di futura ottimizzazione e potenziamento degli strumenti open-source, bisogna annotare come tutti gli elementi facente parte del flusso OPENRoad, dai vari script utilizzati, ai codici descrittivi dei FOSS facenti parte del flusso, fino all'annessa documentazione siano in constante sviluppo. La pagina github relativa ad OPENRoad è in costante aggiornamento a dismostrazione del fatto che si tratta di un progetto in continuo sviluppo e continuo miglioramento. Di fatto alcuni dei problemi incontrati durante questo lavoro di tesi potrebbero già essere risolti con la prossima release.
In ultimo, rientra pienamente tra le positività dell'ambiente OPENRoad la comunità che lavora dietro al progetto la quale, tramite il forum per sviluppatori *Gitter*, effettua un costante lavoro di supporto ai progettisti che si affacciano al progetto OPENRoad per la prima volta. Durante questo lavoro di tesi l'aiuto ottenuto su tale piattaforma è stato fondametale, dal momento che lì si possono trovare soluzioni ai problemi che l'implementazione di IC tramite OPENRoad può comportare.

6.2 Criticità

Come riportato nei capitoli precedenti, sebbene il flusso OPENRoad goda di importanti potenzialità, esso soffre di alcuni limiti che durante un lavoro di progettazione ed implementazione di IC rischiano di compromettere il lavoro dell'utente che si trova ad utilizzarlo.
Come analizzato nel capitolo relativo alla sintesi dell'architettura sotto analisi, il FOSS utilizzato nel flusso OPENRoad, Yosys/ABC, non è in grado di effettuare modifiche sostanziali nella netlist generata che tengano conto della frequenza operativa scelta. Ciò si traduce in una mancatà ottimizzazione della netlist ed in consequenti penalità in termini soprattutto di timing e potenza dissipata.
Altro limite di un certo rilievo risiede nel successivo step di floorplan. Di fatto il software TritonFP non stima efficacemente l'area da destinare al design utilizzando i tipici parametri di CORE_UTILIZZATION e PLACE_DENSITY, capaci quindi di rendere tale passo della fase di P&R il più versatile possibile. Il progettista deve in questo caso indicare le effettive dimensioni del design, per di più sovrastimandole, per evitare errori in fase di routing. Questo vincolo va in realtà contro uno dei principi cardine del progetto del DARPA, citato precedentemente, ovvero il "no human in the loop". In questo caso la "presenza umana" deve essere consistente poichè l'utente deve continuamente intervenire nel processo di P&R proprio per evitare errori durante tutto il flusso, errori dovuti a stime approssimative nella fase di inizializzazione del flusso.
Ultima criticità strettamente legata ad i tool presenti in ambiente open-source riguarda, come detto nel corrispettivo capitolo, l'uso di buffer o inverter per la realizzazione dell'albero di clock. In questo caso il FOSS TritonCTS, al di là di inserire in modo indiscriminato tali elimenti, sia che effettivamente la struttura sotto analisi ne abbia bisogno o meno (si pensi al caso degli elementi più semplici come NV_NVDLA_reset o

NV_NVDLA_sync3d), non è in grado di discernere all'interno di una lista di buffer ed inverter quali possano effettivamente apportare un contributo positivo in termini di timing o potenza dissipata. Esso si limita alla scelta di un unico elemento, nella fattispecie il primo indicato nella lista di possibili celle utilizzabili. Risulta chiaro come questo risulti essere un importante limite in una visione di ottimizzazione del design da implementare.
Come più volte riportato, durante questo lavoro di tesi ci siamo spesso ritrovati costretti a ridurre le entità analizzate, specialmente in fase di P&R, poichè limitati dall'eccessivo carico computazionale che gravava sul sistema dato dall'implementazione di questi elementi. Questo è sicuramente un grosso limite del flusso OPENRoad. In un contesto in cui le complessità dei circuiti integrati è in continuo aumento gli strumenti di sintesi e P&R devono essere in grado di poter gestire al meglio tale crescita. In questo l'ambiente open-source è certamente lacunoso.
Ultimo elemento riportato come criticità del flusso OPENRoad riguarda i tempi di eseguzione di un implementazione completa, dalla sintesi alla generazione del file GDSII, di un blocco di una discreta complessità quale NV_NVDLA_CACC_regfile.
L'estratto in basso mostra un l'ammontare di ore di computazione per completare un flusso di sintesi a generazione file GDSII. Si può vedere come l'implementazione di questa instanza abbia richiesto all'incirca 300 ore di computazione, ovvero circa 12 giorni di continuo lavoro da parte del sistema. Risulta ovvio intuire come tale intervallo termporale sia estremamente grande, cosiderando anche che l'elemento ha una medio-bassa complessità.

Report 7 Estratto del file di log in cui è evidenziato il tempo di P&R trascorso

```
[H]
TritonRoute.log:[INFO DRT-0172]  elapsed time = 272:41:35
```

In conclusione a questo lavoro di tesi ci sentiamo di poter garantire quanto il progetto OPENRoad abbia tutte le caratteristiche di diventare uno strandard nella realizzazione ed implementazione di IC, portando a compimento quel processo di "democratizzazione dell'ingegneria elettronica", che è uno dei principi ispiranti del progetto.
Purtroppo ad oggi non è ancora in grado di garantire la sicurezza e la stabilità dei risultati se confrontati con i vari software proprietari utilizzati durante l'attività lavorativa del progettista di circuiti integrati.
La speranza è che riesca a passare attraverso un'intensa attività di ricerca e sviluppo in grado di colmare, se non tutti, una parte di quelle limitazioni che lo rendono ancora troppo distante dai competitor attualmente utilizzati, quali sono gli strumenti della suite Cadence.

Appendice A

NVDLA: script di configurazione

FILE DI CONFIGURAZIONE: CONFIG.SH

```
    # ================================================================
# File: syn/templates/config.sh
# NVDLA Open Source Project
# Template configuration file for reference synthesis methodology
#
# Copyright (c) 2016 - 2017 NVIDIA Corporation. Licensed under the
# NVDLA Open Hardware License; see the "LICENSE.txt" file that came
# with this distribution for more information.
# ================================================================

# ============================
# DESIGN RELATED VARIABLES
# ============================

export TOP_NAMES="NV_NVDLA_partition_a_netlist"

export NVDLA_ROOT="/home/thesis/antonino.magaddino/hw"

# Where do I find the RTL source verilog/system verilog files?
export RTL_SEARCH_PATH="$(ls -d ${NVDLA_ROOT}/vmod/nvdla/*) \
${NVDLA_ROOT}/vmod/vlibs \
    ${NVDLA_ROOT}/vmod/rams/synth \
    ${NVDLA_ROOT}/vmod/rams/model \
"

# For verilog source files that do not match the module name.
export EXTRA_RTL="${NVDLA_ROOT}/vmod/nvdla/nocif/NV_NVDLA_XXIF_libs.v"

# If there are verilog header files, where do I find them?
export RTL_INCLUDE_SEARCH_PATH=" \
```

```
    ${NVDLA_ROOT}/vmod/include \
"


# File extensions for source files...
export RTL_EXTENSIONS=".v .sv .gv"
export RTL_INCLUDE_EXTENSIONS=".vh .svh"

# FLoorplans and constraints
export DEF="def"
export CONS="/home/thesis/antonino.magaddino/hw/syn/cons"

# ============================
# TOOL RELATED VARIABLES
# ============================


# Design Compiler Installation - Where do I find the dc_shell executable
export DC_PATH="/software/europractice-release-2019/synopsys/syn0-2018.06-SP4/bin"


# ============================
# LIBRARY RELATED VARIABLES
# ============================

export RELEASE_DIR="/home/thesis/antonino.magaddino/skywater-pdk/libraries/
                    /sky130_fd_sc_hd/latest/timing"
#export RAM_LIB_DIR=""
export TARGET_LIB="${RELEASE_DIR}/sky130_fd_sc_hd__tt_025C_1v80.db"
export LINK_LIB="${RELEASE_DIR}/sky130_fd_sc_hd__tt_025C_1v80.db \
    ${DC_PATH}/../libraries/syn/dw_foundation.sldb \
    ${DC_PATH}/../libraries/syn/gtech.db \
    ${DC_PATH}/../libraries/syn/standard.sldb \
"

#export MW_LIB=""

#export TF_FILE=""
#export TLUPLUS_FILE=""
#export TLUPLUS_MAPPING_FILE=""
export MIN_ROUTING_LAYER="lil"
export MAX_ROUTING_LAYER="met5"
export HORIZONTAL_LAYERS="met1 met3 met5"
export VERTICAL_LAYERS="lil met2 met4"
```

```
#export WIRELOAD_MODEL_NAME=""
#export WIRELOAD_MODEL_FILE=""
export DONT_USE_LIST=""

# ==========================
# MISCELLANEOUS VARIABLES
#==========================
# Set host options in the DC session.
export DC_NUM_CORES="8"

# Apply constraints to tighten CG enable paths to model post-CTS insertion delays
export TIGHTEN_CGE="1"

# Enable Area recovery (run optimize_netlist -area)
export AREA_RECOVERY="1"

# Number of incremental recompile loops
export INCREMENTAL_RECOMPILE_COUNT="2"

# For Job management
export COMMAND_PREFIX=""

export CGLUT_FILE="$NVDLA_ROOT/templates/cg_latency_lut.tcl"
```

SCRIPT DI SINTESI: SYN_LAUNCH.SH

```
    #!/usr/bin/bash
# ================================================================
# File: syn/scripts/syn_launch.sh
# NVDLA Open Source Project
# Control script for the reference synthesis methodology
#
# Copyright (c) 2016 - 2017 NVIDIA Corporation. Licensed under the
# NVDLA Open Hardware License; see the "LICENSE.txt" file that came
# with this distribution for more information.
# ================================================================

# Help function
usage ()
{
    echo Usage: $0 \[-build STRING\] \[-config /path/to/config\]
                -mode \[STRING\] -restore \[STRING\];
    exit 1;
```

```
}

# Configure defaults
FLOW_ROOT=`dirname $0`
DEFAULT_FLOW_CONFIG=$FLOW_ROOT/default_config.sh

# Set up defaults.
timestamp=$(date +%Y%m%d_%H%M)
config="./config.sh"
mode="wlm"
build="nvdla_syn_$timestamp"
modules=""
restore_db=""
qa_mode=""

while [ $# -gt 0 ]
do
  case $1 in
      -config)
          error=0
          shift
          config="$1" ;;
      -mode)
          error=0
          shift
          mode="$1" ;;
      -modules)
          error=0
          shift
          modules="$1" ;;
      -build)
          error=0
          shift
          build="$1" ;;
      -restore)
          error=0
          shift
          restore_db="$1" ;;
      -qa_mode)
          error=0
          shift
          qa_mode="$1" ;;
        *)
          echo Error: unrecognized argument: $1
```

```
          usage ;;
    esac
  shift
done

echo "[INFO]: Sourcing default flow variables from $DEFAULT_FLOW_CONFIG ... "
source $DEFAULT_FLOW_CONFIG

# Source config file
if [ ! -f "$config" ] ; then
    echo "[ERROR]: Please provide a valid config file."
    usage
fi
echo "[INFO]: Sourcing user synthesis configuration file $config ... "
source $config

# enable license queuing
export SNPSLMD_QUEUE=true

# If user is running QA mode, then pass that on to the TCL scripts
export QA_MODE="$qa_mode"

if [ -z "$modules" ] && [ -z "$TOP_NAMES" ] ; then
    echo "[ERROR]: TOP_NAMES cannot be empty. Aborting"
    exit
elif [ -z "$modules" ] ; then
    modules=$TOP_NAMES
fi

export BUILD_NAME=$build
export DB_DIR="$BUILD_NAME/db"
export CONS_DIR="$BUILD_NAME/cons"
export DEF_DIR="$BUILD_NAME/def"
export DLIB_DIR="$BUILD_NAME/design_lib"
export FV_DIR="$BUILD_NAME/fv"
export LOG_DIR="$BUILD_NAME/log"
export MW_DIR="$BUILD_NAME/mw"
export NET_DIR="$BUILD_NAME/net"
export REPORT_DIR="$BUILD_NAME/report"
export SCRIPTS_DIR="$BUILD_NAME/scripts"
export SEARCH_PATH=". $BUILD_NAME/src"
# Helper function to create BUILD sandbox
dataprep()
{
```

```
    if [ -d  $BUILD_NAME ] ; then
        echo "[INFO]: Cleaning up previous build directory $BUILD_NAME..."
        rm -rf $BUILD_NAME
    fi
    if [ ! -d "$BUILD_NAME" ] ; then
      echo "[INFO]: Creating sandbox $BUILD_NAME ... "
    mkdir -p $BUILD_NAME
        mkdir -p $BUILD_NAME/cons
        mkdir -p $BUILD_NAME/log
        mkdir -p $BUILD_NAME/report
        mkdir -p $BUILD_NAME/db
        mkdir -p $BUILD_NAME/scripts
        mkdir -p $BUILD_NAME/design_lib
     mkdir -p $BUILD_NAME/mw
     mkdir -p $BUILD_NAME/fv
mkdir -p $BUILD_NAME/def
mkdir -p $BUILD_NAME/net
     mkdir -p $BUILD_NAME/src
for module in $modules
do
     mkdir -p $BUILD_NAME/fv/${module}
done

        echo "[INFO]: Copying flow source code into $BUILD_NAME/scripts/ ..."
       # cp -Lrf ${FLOW_ROOT}/* $BUILD_NAME/scripts/
 cp -Lf ${FLOW_ROOT}/* $BUILD_NAME/scripts/


        if [ "$DEF" != "" ] && [ -d "$DEF" ] ; then
            echo "[INFO]: Copying DEF files if available, into $BUILD_NAME/def..."
            cp -Lrf $DEF/* $BUILD_NAME/def/
        fi
        if [ "$CONS" != "" ] && [ -d "$CONS" ] ; then
            echo "[INFO]: Copying constraint files if available,
                          into $BUILD_NAME/cons ..."
            cp -Lrf $CONS/* $BUILD_NAME/cons/
        fi
    fi

    echo "[INFO]: Searching for RTL with extension: $RTL_EXTENSIONS "
for path in ${RTL_SEARCH_PATH}
    do
    for ext in ${RTL_EXTENSIONS}
        do
```

```
     cp -Lrf $path/*$ext $BUILD_NAME/src/ >& /dev/null
        done
    done

    echo "[INFO]: Searching for INCLUDE files with extension: $RTL_INCLUDE_EXTENSIONS "
for path in ${RTL_INCLUDE_SEARCH_PATH}
    do
        for ext in ${RTL_INCLUDE_EXTENSIONS}
        do
     cp -Lrf $path/*$ext $BUILD_NAME/src/ >& /dev/null
        done
    done

    EXTRA_RTL_LIST=""
    for file in ${EXTRA_RTL}
    do
        cp -Lrf $file $BUILD_NAME/src
        FILE_NAME=$(basename $file)
        EXTRA_RTL_LIST+=$BUILD_NAME/src/$FILE_NAME
    done
    echo "[INFO]: Copied all RTL and include files into $BUILD_NAME/src"


    echo "[INFO]: Removing any designware components from $BUILD_NAME/src"
    DW_FILES=$(ls $BUILD_NAME/src/DW_*)
    if [ ! -z "$DW_FILES" ] ; then
        rm -rf $BUILD_NAME/src/DW_*
    fi


    for module in $modules
do
        rm -rf $BUILD_NAME/scripts/${module}.files.vc
        echo "-y $BUILD_NAME/src" > $BUILD_NAME/scripts/${module}.files.vc
        echo "+incdir+$BUILD_NAME/src" >> $BUILD_NAME/scripts/${module}.files.vc
        for ext in ${RTL_EXTENSIONS}
        do
            echo "+libext+$ext" >> $BUILD_NAME/scripts/${module}.files.vc
        done

        echo "+define+DISABLE_TESTPOINTS" >> $BUILD_NAME/scripts/${module}.files.vc
        echo "+define+NV_SYNTHESIS " >> $BUILD_NAME/scripts/${module}.files.vc
        echo "+define+RAM_INTERFACE " >> $BUILD_NAME/scripts/${module}.files.vc
        echo "$module.v" >> $BUILD_NAME/scripts/${module}.files.vc
```

```
        # Common "library" modules
        for file in $EXTRA_RTL_LIST
        do
            echo "-v $file" >> $BUILD_NAME/scripts/${module}.files.vc
        done
        echo "[INFO]: Generated module input dependency file $BUILD_NAME/scripts/
                      /${module}.files.vc"

done
}

# Run the data prep stage.
if [ -z "$restore_db" ] ; then
    dataprep
fi


if [ -z "$DC_PATH" ] ; then
    echo "[ERROR]: DC_PATH cannot be empty. Aborting"
    exit 1
fi

if [ "$mode" == "dct" ] || [ "$mode" == "dcg"  ] ; then
echo "[INFO]: Running DC - Topographical..."
export SYN_MODE=$mode
for module in $modules
do
export MODULE=$module
        export RTL_DEPS="$BUILD_NAME/scripts/${module}.files.vc"
        COMMAND_PREFIX_PATCHED="${COMMAND_PREFIX/\<MODULE\>/$module}"
        COMMAND_PREFIX_PATCHED="${COMMAND_PREFIX_PATCHED/\<LOG\>/$LOG_DIR}"
        if [ -z "$restore_db" ] ; then
            echo $COMMAND_PREFIX_PATCHED $DC_PATH/dc_shell-t -topographical_mode
            -no_gui -f $BUILD_NAME/scripts/dc_run.tcl -output_log_file
            $LOG_DIR/${module}_${SYN_MODE}.log
            $COMMAND_PREFIX_PATCHED $DC_PATH/dc_shell-t -topographical_mode
            -no_gui -f $BUILD_NAME/scripts/dc_run.tcl -output_log_file
            $LOG_DIR/${module}_${SYN_MODE}.log
        else
            export RESTORE_DB=$restore_db
            echo $COMMAND_PREFIX_PATCHED $DC_PATH/dc_shell-t -topographical_mode
            -f $BUILD_NAME/scripts/dc_interactive.tcl -output_log_file
            $LOG_DIR/${module}_${SYN_MODE}.interactive.log
```

```
            $COMMAND_PREFIX_PATCHED $DC_PATH/dc_shell-t -topographical_mode
            -f $BUILD_NAME/scripts/dc_interactive.tcl -output_log_file
            $LOG_DIR/${module}_${SYN_MODE}.interactive.log
        fi
done
elif [ "$mode" == "wlm" ] ; then
    echo "[INFO]: Running DC (non-physical/Wireload model)..."
export SYN_MODE=$mode
for module in $modules
do
export MODULE=$module
        export RTL_DEPS="$BUILD_NAME/scripts/${module}.files.vc"
        COMMAND_PREFIX_PATCHED="${COMMAND_PREFIX/\<MODULE\>/$module}"
        COMMAND_PREFIX_PATCHED="${COMMAND_PREFIX_PATCHED/\<LOG\>/$LOG_DIR}"
        if [ -z "$restore_db" ] ; then
    echo "starting dc_shell"
            echo $COMMAND_PREFIX_PATCHED $DC_PATH/dc_shell -no_gui
            -f $BUILD_NAME/scripts/dc_run.tcl -output_log_file
            $LOG_DIR/${module}_${SYN_MODE}.log
    echo "ending dc_shell"
            $COMMAND_PREFIX_PATCHED $DC_PATH/dc_shell -no_gui
            -f $BUILD_NAME/scripts/dc_run.tcl -output_log_file
            $LOG_DIR/${module}_${SYN_MODE}.log
        else
            export RESTORE_DB=$restore_db
            echo $COMMAND_PREFIX_PATCHED $DC_PATH/dc_shell-t -f $BUILD_NAME/scripts/
            /dc_interactive.tcl -output_log_file $LOG_DIR/
            /${module}_${SYN_MODE}.interactive.log
            $COMMAND_PREFIX_PATCHED $DC_PATH/dc_shell-t -f $BUILD_NAME/scripts/
            /dc_interactive.tcl -output_log_file $LOG_DIR/
            /${module}_${SYN_MODE}.interactive.log
        fi
    done
elif [ "$mode" == "de" ] ; then
    echo "[INFO] Running Design Explorer ..."
export SYN_MODE=$mode
for module in $modules
do
export MODULE=$module
        export RTL_DEPS="$BUILD_NAME/scripts/${module}.files.vc"
        COMMAND_PREFIX_PATCHED="${COMMAND_PREFIX/\<MODULE\>/$module}"
        COMMAND_PREFIX_PATCHED="${COMMAND_PREFIX_PATCHED/\<LOG\>/$LOG_DIR}"
        if [ -z "$restore_db" ] ; then
    echo $COMMAND_PREFIX_PATCHED $DC_PATH/de_shell -no_gui -f $BUILD_NAME/scripts/
```

```
    /dc_run.tcl -output_log_file $LOG_DIR/${module}_${SYN_MODE}.interactive.log
     $COMMAND_PREFIX_PATCHED $DC_PATH/de_shell -no_gui -f $BUILD_NAME/scripts/dc_run.tcl
        else
            export RESTORE_DB=$restore_db
    echo $COMMAND_PREFIX_PATCHED $DC_PATH/de_shell  -f $BUILD_NAME/scripts/
    /dc_interactive.tcl -output_log_file $LOG_DIR/${module}_${SYN_MODE}.interactive.log
    $COMMAND_PREFIX_PATCHED $DC_PATH/de_shell -f $BUILD_NAME/scripts/
    /dc_interactive.tcl -output_log_file $LOG_DIR/${module}_${SYN_MODE}.interactive.log
        fi
    done
else
    echo "[ERROR]: Unsupported option for -mode - Only supported modes are:
    \"wlm,dct,dcg,de\". Please refer the documentation for more details"
    exit 1
fi
```

Appendice B

NVDLA: codici verilog

ESEMPIO RAM SINTETIZABLIE: RAMDP_16X256_GL_M2_E2.v

```
// ========================================================================
// NVDLA Open Source Project
// Copyright(c) 2016 - 2017 NVIDIA Corporation. Licensed under the
// NVDLA Open Hardware License; Check "LICENSE" which comes with
// this distribution for more information.
// ========================================================================
// File Name: RAMDP_16X256_GL_M2_E2.v

`timescale 10ps/1ps
module RAMDP_16X256_GL_M1_E2 (CLK_R, CLK_W, RE, WE, RADR, WADR, WD,
RD, IDDQ, SVOP, SLEEP_EN, RET_EN);
input CLK_R,CLK_W,WE,RE,IDDQ,RET_EN;
input [3:0] RADR,WADR;
input [7:0] SLEEP_EN;
input [1:0] SVOP;
input [255:0] WD;
output [255:0] RD;
reg [255:0] ram [15:0];
reg [255:0] RD;

always @(posedge CLK_W) begin
  if (WE)
      ram[WADR] <= WD;
end

always @(posedge CLK_R) begin
  if (RE)
    RD <= ram[RADR];
end
endmodule
```

Appendice C

Critical path di sintesi e PR

Critical Path per la netlist generata da Yosys/ABC ed analizzata da Synopsys Design Compiler ad una frequenza operativa di 200MHz

```
Startpoint: _1125085_ (rising edge-triggered flip-flop clocked by nvdla_core_clk)
Endpoint: _1058173_ (rising edge-triggered flip-flop clocked by nvdla_core_clk)
Path Group: nvdla_core_clk
Path Type: max

 Point                                          Fanout      Incr       Path
 ---------------------------------------------------------------------------
clock nvdla_core_clk (rise edge)                          0.0000     0.0000
clock network delay (ideal)                               0.0000     0.0000
_1125085_/CLK (sky130_fd_sc_hd__dfrtp_1)                  0.0000     0.0000 r
_1125085_/Q (sky130_fd_sc_hd__dfrtp_1)                    0.3642     0.3642 f
u_assembly_ctrl.cfg_truncate[102] (net)        1          0.0000     0.3642 f
_0524619_/A (sky130_fd_sc_hd__buf_6)                      0.0000     0.3642 f
_0524619_/X (sky130_fd_sc_hd__buf_6)                      0.1384     0.5026 f
_0084002_ (net)                                9          0.0000     0.5026 f
_0524690_/A (sky130_fd_sc_hd__inv_2)                      0.0000     0.5026 f
_0524690_/Y (sky130_fd_sc_hd__inv_2)                      0.0735     0.5762 r
_0084073_ (net)                                3          0.0000     0.5762 r
_0524907_/A (sky130_fd_sc_hd__clkbuf_8)                   0.0000     0.5762 r
_0524907_/X (sky130_fd_sc_hd__clkbuf_8)                   0.1633     0.7394 r
_0084290_ (net)                               10          0.0000     0.7394 r
_0526717_/A (sky130_fd_sc_hd__buf_12)                     0.0000     0.7394 r
_0526717_/X (sky130_fd_sc_hd__buf_12)                     0.1182     0.8577 r
_0086098_ (net)                               10          0.0000     0.8577 r
_0526718_/A (sky130_fd_sc_hd__clkbuf_8)                   0.0000     0.8577 r
_0526718_/X (sky130_fd_sc_hd__clkbuf_8)                   0.1403     0.9980 r
_0086099_ (net)                               10          0.0000     0.9980 r
_0528762_/A (sky130_fd_sc_hd__buf_2)                      0.0000     0.9980 r
_0528762_/X (sky130_fd_sc_hd__buf_2)                      0.1773     1.1753 r
```

```
_0088137_ (net)                          10       0.0000     1.1753 r
_0528775_/A (sky130_fd_sc_hd__buf_4)              0.0000     1.1753 r
_0528775_/X (sky130_fd_sc_hd__buf_4)              0.1793     1.3546 r
_0088150_ (net)                          10       0.0000     1.3546 r
_0528886_/A (sky130_fd_sc_hd__buf_8)              0.0000     1.3546 r
_0528886_/X (sky130_fd_sc_hd__buf_8)              0.1195     1.4741 r
_0088261_ (net)                          10       0.0000     1.4741 r
_0528887_/A (sky130_fd_sc_hd__buf_4)              0.0000     1.4741 r
_0528887_/X (sky130_fd_sc_hd__buf_4)              0.1423     1.6164 r
_0088262_ (net)                          10       0.0000     1.6164 r
_0528888_/A (sky130_fd_sc_hd__buf_4)              0.0000     1.6164 r
_0528888_/X (sky130_fd_sc_hd__buf_4)              0.1542     1.7706 r
_0088263_ (net)                          10       0.0000     1.7706 r
_0534563_/A (sky130_fd_sc_hd__buf_2)              0.0000     1.7706 r
_0534563_/X (sky130_fd_sc_hd__buf_2)              0.1695     1.9401 r
_0093918_ (net)                          10       0.0000     1.9401 r
_0534564_/A (sky130_fd_sc_hd__buf_2)              0.0000     1.9401 r
_0534564_/X (sky130_fd_sc_hd__buf_2)              0.1864     2.1264 r
_0093919_ (net)                          10       0.0000     2.1264 r
_0534912_/A (sky130_fd_sc_hd__buf_4)              0.0000     2.1264 r
_0534912_/X (sky130_fd_sc_hd__buf_4)              0.1615     2.2880 r
_0094267_ (net)                          10       0.0000     2.2880 r
_0534913_/A (sky130_fd_sc_hd__buf_2)              0.0000     2.2880 r
_0534913_/X (sky130_fd_sc_hd__buf_2)              0.1680     2.4560 r
_0094268_ (net)                          10       0.0000     2.4560 r
_0562802_/A (sky130_fd_sc_hd__buf_2)              0.0000     2.4560 r
_0562802_/X (sky130_fd_sc_hd__buf_2)              0.1929     2.6488 r
_0122055_ (net)                          10       0.0000     2.6488 r
_0566431_/A (sky130_fd_sc_hd__buf_2)              0.0000     2.6488 r
_0566431_/X (sky130_fd_sc_hd__buf_2)              0.1855     2.8344 r
_0125674_ (net)                          10       0.0000     2.8344 r
_0566432_/A (sky130_fd_sc_hd__buf_2)              0.0000     2.8344 r
_0566432_/X (sky130_fd_sc_hd__buf_2)              0.1861     3.0205 r
_0125675_ (net)                          10       0.0000     3.0205 r
_0566433_/A (sky130_fd_sc_hd__buf_2)              0.0000     3.0205 r
_0566433_/X (sky130_fd_sc_hd__buf_2)              0.1912     3.2116 r
_0125676_ (net)                          10       0.0000     3.2116 r
_0575936_/A (sky130_fd_sc_hd__buf_2)              0.0000     3.2116 r
_0575936_/X (sky130_fd_sc_hd__buf_2)              0.2021     3.4137 r
_0135157_ (net)                          10       0.0000     3.4137 r
_0586959_/S (sky130_fd_sc_hd__mux2i_1)            0.0000     3.4137 r
_0586959_/Y (sky130_fd_sc_hd__mux2i_1)            0.2133     3.6270 r
_0146162_ (net)                           2       0.0000     3.6270 r
_0586960_/A2 (sky130_fd_sc_hd__o21ai_0)           0.0000     3.6270 r
```

```
_0586960_/Y (sky130_fd_sc_hd__o21ai_0)              0.1035     3.7305 f
_0146163_ (net)                                2    0.0000     3.7305 f
_0586962_/A2 (sky130_fd_sc_hd__o21ai_0)             0.0000     3.7305 f
_0586962_/Y (sky130_fd_sc_hd__o21ai_0)              0.2433     3.9738 r
_0146165_ (net)                                2    0.0000     3.9738 r
_0586963_/A (sky130_fd_sc_hd__inv_1)                0.0000     3.9738 r
_0586963_/Y (sky130_fd_sc_hd__inv_1)                0.0982     4.0721 f
_0146166_ (net)                                3    0.0000     4.0721 f
_0586983_/A (sky130_fd_sc_hd__nor4_1)               0.0000     4.0721 f
_0586983_/Y (sky130_fd_sc_hd__nor4_1)               0.2616     4.3336 r
_0146186_ (net)                                1    0.0000     4.3336 r
_0587023_/C (sky130_fd_sc_hd__nand4_1)              0.0000     4.3336 r
_0587023_/Y (sky130_fd_sc_hd__nand4_1)              0.1727     4.5064 f
_0146226_ (net)                                3    0.0000     4.5064 f
_0587067_/A2 (sky130_fd_sc_hd__o21ai_2)             0.0000     4.5064 f
_0587067_/Y (sky130_fd_sc_hd__o21ai_2)              0.2728     4.7792 r
_0146270_ (net)                                8    0.0000     4.7792 r
_0587068_/A (sky130_fd_sc_hd__clkinv_1)             0.0000     4.7792 r
_0587068_/Y (sky130_fd_sc_hd__clkinv_1)             0.2276     5.0068 f
_0146271_ (net)                                9    0.0000     5.0068 f
_0695637_/A (sky130_fd_sc_hd__buf_2)                0.0000     5.0068 f
_0695637_/X (sky130_fd_sc_hd__buf_2)                0.2168     5.2236 f
_0242499_ (net)                               10    0.0000     5.2236 f
_0695724_/C (sky130_fd_sc_hd__nand3_1)              0.0000     5.2236 f
_0695724_/Y (sky130_fd_sc_hd__nand3_1)              0.0827     5.3063 r
_0242570_ (net)                                1    0.0000     5.3063 r
_0695726_/B1 (sky130_fd_sc_hd__a22oi_1)             0.0000     5.3063 r
_0695726_/Y (sky130_fd_sc_hd__a22oi_1)              0.0530     5.3593 f
_0012279_ (net)                                1    0.0000     5.3593 f
_1058173_/D (sky130_fd_sc_hd__dfxtp_1)              0.0000     5.3593 f
data arrival time                                              5.3593

clock nvdla_core_clk (rise edge)                    5.0000     5.0000
clock network delay (ideal)                         0.0000     5.0000
_1058173_/CLK (sky130_fd_sc_hd__dfxtp_1)            0.0000     5.0000 r
library setup time                                 -0.1131     4.8869
data required time                                             4.8869
-------------------------------------------------------------------------------
data required time                                             4.8869
data arrival time                                             -5.3593
-------------------------------------------------------------------------------
slack (VIOLATED)                                              -0.4724
```

Critical Path per la netlist generata ed analizzata da Yosys/ABC ad una frequenza operativa di 200MHz

```
Startpoint: _1048884_
            (rising edge-triggered flip-flop clocked by nvdla_core_clk)
Endpoint: _1048785_ (rising edge-triggered flip-flop clocked by nvdla_core_clk)
Path Group: nvdla_core_clk
Path Type: max

Fanout     Cap    Slew   Delay    Time   Description
---------------------------------------------------------------------------
                  0.05    0.00    0.00   clock nvdla_core_clk (rise edge)
                          0.00    0.00   clock network delay (ideal)
                  0.05    0.00    0.00 ^ _1048884_/CLK (sky130_fd_sc_hd__dfxtp_1)
                  0.11    0.36    0.36 v _1048884_/Q (sky130_fd_sc_hd__dfxtp_1)
     5    0.02                           u_calculator.u_cell_int_112.i_partial_result[33]
                  0.11    0.01    0.37 v _0535568_/A (sky130_fd_sc_hd__inv_2)
                  0.18    0.18    0.55 ^ _0535568_/Y (sky130_fd_sc_hd__inv_2)
     8    0.04                           _0094917_ (net)
                  0.18    0.01    0.56 ^ _0535569_/B (sky130_fd_sc_hd__nor2_4)
                  0.05    0.06    0.62 v _0535569_/Y (sky130_fd_sc_hd__nor2_4)
     4    0.02                           _0094918_ (net)
                  0.06    0.01    0.63 v _0535574_/A (sky130_fd_sc_hd__inv_4)
                  0.06    0.07    0.70 ^ _0535574_/Y (sky130_fd_sc_hd__inv_4)
     3    0.02                           _0094923_ (net)
                  0.06    0.01    0.70 ^ _0535631_/A (sky130_fd_sc_hd__buf_8)
                  0.09    0.13    0.84 ^ _0535631_/X (sky130_fd_sc_hd__buf_8)
    10    0.05                           _0094980_ (net)
                  0.09    0.00    0.84 ^ _0535734_/B1 (sky130_fd_sc_hd__o21ai_2)
                  0.04    0.06    0.90 v _0535734_/Y (sky130_fd_sc_hd__o21ai_2)
     2    0.00                           _0095083_ (net)
                  0.04    0.00    0.90 v _0535735_/B (sky130_fd_sc_hd__or2_1)
                  0.06    0.21    1.12 v _0535735_/X (sky130_fd_sc_hd__or2_1)
     2    0.01                           _0095084_ (net)
                  0.06    0.00    1.12 v _0535736_/A (sky130_fd_sc_hd__nand2_1)
                  0.04    0.06    1.17 ^ _0535736_/Y (sky130_fd_sc_hd__nand2_1)
     1    0.00                           _0095085_ (net)
                  0.04    0.00    1.18 ^ _0535737_/B1 (sky130_fd_sc_hd__o21ai_1)
                  0.06    0.06    1.24 v _0535737_/Y (sky130_fd_sc_hd__o21ai_1)
     2    0.00                           _0095086_ (net)
                  0.06    0.00    1.25 v _0535772_/A0 (sky130_fd_sc_hd__mux2i_1)
                  0.16    0.16    1.40 ^ _0535772_/Y (sky130_fd_sc_hd__mux2i_1)
     2    0.01                           _0095121_ (net)
```

```
              0.16    0.00    1.41 ^ _0535811_/A (sky130_fd_sc_hd__nand2_1)
              0.06    0.08    1.49 v _0535811_/Y (sky130_fd_sc_hd__nand2_1)
1     0.00                           _0095160_ (net)
              0.07    0.01    1.49 v _0535812_/B1 (sky130_fd_sc_hd__o21ai_2)
              0.13    0.07    1.57 ^ _0535812_/Y (sky130_fd_sc_hd__o21ai_2)
2     0.01                           _0095161_ (net)
              0.13    0.01    1.57 ^ _0535813_/A (sky130_fd_sc_hd__inv_2)
              0.03    0.04    1.62 v _0535813_/Y (sky130_fd_sc_hd__inv_2)
1     0.00                           _0095162_ (net)
              0.04    0.01    1.62 v _0535817_/A2 (sky130_fd_sc_hd__o21ai_2)
              0.10    0.11    1.73 ^ _0535817_/Y (sky130_fd_sc_hd__o21ai_2)
2     0.00                           _0095166_ (net)
              0.10    0.00    1.74 ^ _0535993_/B (sky130_fd_sc_hd__nor2_1)
              0.03    0.05    1.79 v _0535993_/Y (sky130_fd_sc_hd__nor2_1)
1     0.00                           _0095342_ (net)
              0.03    0.00    1.79 v _0535997_/A (sky130_fd_sc_hd__nand3_1)
              0.06    0.06    1.85 ^ _0535997_/Y (sky130_fd_sc_hd__nand3_1)
1     0.00                           _0095346_ (net)
              0.06    0.00    1.85 ^ _0535998_/B (sky130_fd_sc_hd__nor2_1)
              0.03    0.04    1.89 v _0535998_/Y (sky130_fd_sc_hd__nor2_1)
1     0.00                           _0095347_ (net)
              0.03    0.00    1.90 v _0536008_/A2 (sky130_fd_sc_hd__a31oi_1)
              0.13    0.15    2.04 ^ _0536008_/Y (sky130_fd_sc_hd__a31oi_1)
1     0.00                           _0095357_ (net)
              0.13    0.00    2.05 ^ _0536009_/B (sky130_fd_sc_hd__nor2_1)
              0.04    0.05    2.10 v _0536009_/Y (sky130_fd_sc_hd__nor2_1)
1     0.00                           _0095358_ (net)
              0.04    0.00    2.10 v _0536010_/B (sky130_fd_sc_hd__nor2_1)
              0.14    0.12    2.23 ^ _0536010_/Y (sky130_fd_sc_hd__nor2_1)
2     0.01                           _0095359_ (net)
              0.14    0.01    2.23 ^ _0536015_/A (sky130_fd_sc_hd__nand2_2)
              0.08    0.09    2.32 v _0536015_/Y (sky130_fd_sc_hd__nand2_2)
3     0.01                           _0095364_ (net)
              0.08    0.01    2.33 v _0536016_/B (sky130_fd_sc_hd__nor2_4)
              0.10    0.12    2.45 ^ _0536016_/Y (sky130_fd_sc_hd__nor2_4)
3     0.01                           _0095365_ (net)
              0.11    0.01    2.46 ^ _0536019_/A (sky130_fd_sc_hd__nand2_4)
              0.08    0.06    2.52 v _0536019_/Y (sky130_fd_sc_hd__nand2_4)
3     0.01                           _0095368_ (net)
              0.08    0.01    2.53 v _0536020_/B (sky130_fd_sc_hd__nor2_4)
              0.10    0.12    2.65 ^ _0536020_/Y (sky130_fd_sc_hd__nor2_4)
3     0.01                           _0095369_ (net)
              0.11    0.01    2.66 ^ _0536023_/A (sky130_fd_sc_hd__nand2_4)
              0.08    0.06    2.72 v _0536023_/Y (sky130_fd_sc_hd__nand2_4)
```

```
3    0.01                                _0095372_ (net)
             0.08    0.01    2.73 v _0536024_/B (sky130_fd_sc_hd__nor2_4)
             0.10    0.12    2.85 ^ _0536024_/Y (sky130_fd_sc_hd__nor2_4)
3    0.01                                _0095373_ (net)
             0.11    0.01    2.86 ^ _0536028_/A (sky130_fd_sc_hd__nand2_4)
             0.05    0.06    2.92 v _0536028_/Y (sky130_fd_sc_hd__nand2_4)
3    0.01                                _0095377_ (net)
             0.05    0.01    2.93 v _0536029_/B (sky130_fd_sc_hd__nor2_4)
             0.10    0.10    3.04 ^ _0536029_/Y (sky130_fd_sc_hd__nor2_4)
3    0.01                                _0095378_ (net)
             0.11    0.01    3.05 ^ _0536033_/A (sky130_fd_sc_hd__nand2_4)
             0.05    0.06    3.11 v _0536033_/Y (sky130_fd_sc_hd__nand2_4)
3    0.01                                _0095382_ (net)
             0.05    0.01    3.12 v _0536034_/B (sky130_fd_sc_hd__nor2_4)
             0.11    0.11    3.23 ^ _0536034_/Y (sky130_fd_sc_hd__nor2_4)
3    0.01                                _0095383_ (net)
             0.11    0.01    3.24 ^ _0536038_/A (sky130_fd_sc_hd__nand2_4)
             0.07    0.06    3.30 v _0536038_/Y (sky130_fd_sc_hd__nand2_4)
3    0.01                                _0095387_ (net)
             0.07    0.01    3.30 v _0536039_/B (sky130_fd_sc_hd__nor2_2)
             0.17    0.15    3.46 ^ _0536039_/Y (sky130_fd_sc_hd__nor2_2)
3    0.01                                _0095388_ (net)
             0.17    0.01    3.47 ^ _0536042_/A (sky130_fd_sc_hd__nand2_4)
             0.07    0.07    3.54 v _0536042_/Y (sky130_fd_sc_hd__nand2_4)
3    0.01                                _0095391_ (net)
             0.07    0.01    3.55 v _0536043_/B (sky130_fd_sc_hd__nor2_4)
             0.12    0.12    3.67 ^ _0536043_/Y (sky130_fd_sc_hd__nor2_4)
3    0.01                                _0095392_ (net)
             0.12    0.01    3.68 ^ _0536048_/A (sky130_fd_sc_hd__nand2_4)
             0.08    0.07    3.75 v _0536048_/Y (sky130_fd_sc_hd__nand2_4)
3    0.01                                _0095397_ (net)
             0.08    0.01    3.76 v _0536049_/B (sky130_fd_sc_hd__nor2_4)
             0.10    0.11    3.87 ^ _0536049_/Y (sky130_fd_sc_hd__nor2_4)
4    0.01                                _0095398_ (net)
             0.10    0.01    3.88 ^ _0536054_/A (sky130_fd_sc_hd__nand4_2)
             0.14    0.13    4.00 v _0536054_/Y (sky130_fd_sc_hd__nand4_2)
3    0.01                                _0095403_ (net)
             0.14    0.01    4.01 v _0536055_/B (sky130_fd_sc_hd__nor2_4)
             0.10    0.14    4.15 ^ _0536055_/Y (sky130_fd_sc_hd__nor2_4)
2    0.01                                _0095404_ (net)
             0.11    0.01    4.16 ^ _0536059_/A (sky130_fd_sc_hd__nand2_4)
             0.05    0.06    4.22 v _0536059_/Y (sky130_fd_sc_hd__nand2_4)
3    0.01                                _0095408_ (net)
             0.05    0.01    4.23 v _0536060_/B (sky130_fd_sc_hd__nor2_4)
```

```
                0.10    0.10    4.33 ^ _0536060_/Y (sky130_fd_sc_hd__nor2_4)
  2     0.01                            _0095409_ (net)
                0.10    0.01    4.34 ^ _0536065_/A (sky130_fd_sc_hd__nand2_4)
                0.04    0.06    4.40 v _0536065_/Y (sky130_fd_sc_hd__nand2_4)
  3     0.01                            _0095414_ (net)
                0.04    0.01    4.41 v _0536066_/B (sky130_fd_sc_hd__nor2_2)
                0.15    0.13    4.54 ^ _0536066_/Y (sky130_fd_sc_hd__nor2_2)
  2     0.01                            _0095415_ (net)
                0.15    0.01    4.55 ^ _0536071_/A (sky130_fd_sc_hd__nand2_4)
                0.06    0.07    4.62 v _0536071_/Y (sky130_fd_sc_hd__nand2_4)
  4     0.01                            _0095420_ (net)
                0.07    0.01    4.63 v _0536072_/B (sky130_fd_sc_hd__nor2_4)
                0.12    0.12    4.75 ^ _0536072_/Y (sky130_fd_sc_hd__nor2_4)
  3     0.01                            _0095421_ (net)
                0.12    0.01    4.76 ^ _0536074_/A (sky130_fd_sc_hd__nand2_4)
                0.07    0.06    4.82 v _0536074_/Y (sky130_fd_sc_hd__nand2_4)
  3     0.01                            _0095423_ (net)
                0.07    0.01    4.83 v _0536075_/B (sky130_fd_sc_hd__nor2_4)
                0.18    0.17    5.00 ^ _0536075_/Y (sky130_fd_sc_hd__nor2_4)
  5     0.03                            _0095424_ (net)
                0.18    0.01    5.01 ^ _0636817_/A (sky130_fd_sc_hd__nand2_4)
                0.10    0.07    5.08 v _0636817_/Y (sky130_fd_sc_hd__nand2_4)
  3     0.01                            _0193052_ (net)
                0.10    0.01    5.09 v _0636818_/A2 (sky130_fd_sc_hd__o21bai_4)
                0.20    0.21    5.31 ^ _0636818_/Y (sky130_fd_sc_hd__o21bai_4)
  7     0.03                            _0193053_ (net)
                0.20    0.01    5.32 ^ _0636819_/A1 (sky130_fd_sc_hd__a21oi_4)
                0.08    0.10    5.42 v _0636819_/Y (sky130_fd_sc_hd__a21oi_4)
  5     0.02                            _0193054_ (net)
                0.08    0.00    5.42 v _0636820_/A (sky130_fd_sc_hd__buf_4)
                0.07    0.19    5.61 v _0636820_/X (sky130_fd_sc_hd__buf_4)
 10     0.04                            _0193055_ (net)
                0.07    0.00    5.62 v _0636828_/A1 (sky130_fd_sc_hd__a21oi_2)
                0.07    0.11    5.72 ^ _0636828_/Y (sky130_fd_sc_hd__a21oi_2)
  1     0.00                            _0002891_ (net)
                0.07    0.00    5.73 ^ _1048785_/D (sky130_fd_sc_hd__dfxtp_1)
                                5.73   data arrival time

                0.05    5.00    5.00   clock nvdla_core_clk (rise edge)
                        0.00    5.00   clock network delay (ideal)
                        0.00    5.00   clock reconvergence pessimism
                                5.00 ^ _1048785_/CLK (sky130_fd_sc_hd__dfxtp_1)
                       -0.06    4.94   library setup time
                                4.94   data required time
```

```
---------------------------------------------------------------------------
                                4.94   data required time
                               -5.73   data arrival time
---------------------------------------------------------------------------
                               -0.79   slack (VIOLATED)
```

Bibliografia

[1] T. Ajayi, D. Blaauw, T.-B. Chan, C.-K. Cheng, V. A. Chhabria, D. K. Choo, M. Coltella, S. Dobre, R. Dreslinski, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. Li, Z. Liang, U. Mallappa, P. Penzes, G. Pradipta, S. Reda, A. Rovinski, K. Samadi, S. S. Sapatnekar, L. Saul, C. Sechen, V. Srinivas, W. Swartz, D. Sylvester, D. Urquhart, L. Wang, M. Woo and B. Xu, “OpenROAD: Toward a Self-Driving, Open-Source Digital Layout Implementation Tool Chain", Proc. Government Microcircuit Applications and Critical Technology Conference, 2019, pp. 1105-1110.

[2] ABC: A System for Sequential Synthesis and Verification, http://people.eecs.berkeley.edu/ alanmi/abc/.

[3] J. Chen, I. H.-R. Jiang, J. Jung, A. B. Kahng, V. N. Kravets, Y.-L. Li, S.-T. Lin and M. Woo, “DATC RDF-2019: Towards a Complete Academic Reference Design Flow“, Proc. ACM/IEEE International Conference on Computer-Aided Design, 2019, pp. 1-6. (Invited Paper)

[4] T. Ajayi, V. A. Chhabria, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. Lee, U. Mallappa, M. Neseem, G. Pradipta, S. Reda, M. Saligane, S. S. Sapatnekar, C. Sechen, M. Shalan, W. Swartz, L. Wang, Z. Wang, M. Woo and B. Xu, “Toward an Open-Source Digital Flow: First Learnings from the OpenROAD Project“, Proc. ACM/IEEE Design Automation Conference, 2019, pp. 76:1-76:4. (Invited Paper)

[5] C. Chu and Y.-C. Wong, “FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design”, IEEE Trans. on CAD 27(1) (2008), pp. 70-83.

[6] M. Pan, Y. Xu, Y. Zhang, C. Chu, "FastRoute: An efficient and high-quality global router", Proc. Asia and South Pacific Design Automation Conference, 2007, pp. 1-14.

[7] A. B. Kahng, L. Wang and B. Xu, “TritonRoute: An Initial Detailed Router for Advanced VLSI Technologies”, Proc. ICCAD, 2018, pp. 81:1-81:8.

[8] http://nvdla.org/

[9] Ramakrishnan, S. (2020). Implementation of a Deep Learning Inference Accelerator on the FPGA.

[10] https://openram.org/

[11] GUTHAUS, Matthew R., et al. OpenRAM: An open-source memory compiler. In: 2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). IEEE, 2016. p. 1-6.